Titre: Title:	Conception d'architectures de calcul à hautes performances pour la comparaison de séquences génétiques
Auteur: Author:	Nasreddine Hireche
Date:	2008
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Hireche, N. (2008). Conception d'architectures de calcul à hautes performances pour la comparaison de séquences génétiques [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/8255/

UNIVERSITÉ DE MONTRÉAL

CONCEPTION D'ARCHITECTURES DE CALCUL À HAUTES PERFORMANCES POUR LA COMPARAISON DE SÉQUENCES GÉNÉTIQUES

NASREDDINE HIRECHE

DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION DU
DIPLÔME DE MAITRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)

AVRIL 2008

Your file Votre référence
ISBN: 978-0-494-41562-7
Our file Notre référence
ISBN: 978-0-494-41562-7

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

CONCEPTION D'ARCHITECTURES DE CALCUL À HAUTES PERFORMANCES POUR LA COMPARAISON DE SÉQUENCES GÉNÉTIQUES

présenté par : HIRECHE Nasreddine

en vue de l'obtention du diplôme de Maîtrise ès en sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

M. BOIS Guy, Ph.D., président

M. LANGLOIS Pierre, Ph.D., membre et directeur de recherche

Mme NICOLESCU Gabriela, Doct., membre et codirecteur de recherche

M. ABOULHAMID El Mostapha, Ph.D., membre

DÉDICACE

À ma mère et ma grand-mère, mon épouse et tous les membres de ma famille.

REMERCIEMENTS

Je tiens tout d'abord à remercier mon directeur de recherche, M. Pierre Langlois et ma codirectrice Mme Gabriela Nicolescu pour leur encadrement tout au long de ce travail de recherche. Je désire remercier aussi Pierre d'avoir consacré de son temps pour réviser ce mémoire, de m'avoir fourni des remarques pertinentes et constructives.

Je remercie le personnel administratif et technique du département de génie informatique, bien qu'ils n'aient pas participé directement aux travaux de recherche.

Mes remerciements et affections vont à ma chère maman, ma sœur, mes frères, et mes beaux parents, pour leur soutien moral. Un remerciement particulier pour mon frère Mohammed pour ces recommandations et encouragements pleins de persévérance.

Mes vifs remerciements à mon épouse, d'avoir consacré son temps pour lire, réviser et corriger ce mémoire, bien que le mot FPGA n'est qu'un acronyme pour elle.

Merci pour Toufik et son épouse, Youcef, Abdelfattah, Salah, Youssef de Lyon et tous mes amis que je n'ai pas cité ici, pour leurs encouragements et soutien moral qui m'ont incité à accomplir ce mémoire.

RÉSUMÉ

La bioinformatique joue des rôles importants dans la recherche biologique et médicale. La biologie computationnelle et l'organisation des données biologiques se sont reliées aux objectifs du génome visant l'élaboration et la mise en œuvre de stratégies de recherche. L'application de cette information sert à la détermination des origines des maladies, et à d'autres applications commerciales.

La bioinformatique s'intéresse au développement des outils pratiques pour la gestion des données et leur analyse. Les algorithmes utilisés dans ce domaine nécessitent des systèmes à grande puissance de traitement, d'où le défi de calcul à haute performance de ces applications.

La loi de Moore montre que le nombre de transistors intégrables double tous les 18 mois. Cependant, la quantité de données génomiques dans *GenBank*, une collection annotée de séquences d'ADN disponible publiquement, double tous les six mois. Le fossé entre la quantité de l'information à analyser et la densité d'intégration ne cesse de croître. Ceci implique une mise en œuvre de différentes stratégies d'implémentation. Un effort considérable de conception est requis pour choisir le meilleur compromis entre trois critères: flexibilité, programmabilité et densité computationnelle du système de traitement à réaliser.

L'objectif ultime de ce projet est d'apporter un gain de performances à l'application de comparaison de séquences génétiques. Principalement, ce gain provient de l'accélération de la vitesse de traitement, et l'augmentation de la capacité de comparer des séquences génétiques plus longues.

Pour accélérer la vitesse de traitement, deux méthodes de parallélisation sont adoptées. D'abord, une parallélisation logicielle est effectuée. Elle utilise des approches basées sur les standards de programmation parallèle OpenMP (*Open Multi-Processing*)

et MPI (Message-Passing Interface). En suite, une méthode de parallélisation matérielle est suivie. Elle se résume à une étude de conception et des réalisations de systèmes de traitement. Une simplification des formules de calcul du résultat de la programmation dynamique a apporté des gains à la vitesse de traitement, ainsi qu'au coût matériel des systèmes réalisés. Ces systèmes ont des niveaux de granularité différents, simple, double et quadruple. Ceci a donné lieu à une étude comparative de leurs performances. Ces systèmes de traitement ont permis d'obtenir une complexité $O(n)$, au lieu de $O(n^2)$ dans le cas d'un algorithme de programmation dynamique implémenté en logiciel.

Notre tableau systolique basé sur un élément de traitement simple (*PE_1×1*) présente une accélération qui varie entre 13% et 142% comparativement aux réalisations existantes. À titre indicatif, notre *PE_1×1×7400* peut comparer un ensemble de 3.54 millions de séquences ayant une longueur de 7400 nucléotides, durant 1 minute uniquement. Ce tableau systolique a la plus grande vitesse de traitement pour la plus grande taille des séquences (7400 nucléotides) qu'on peut implémenter sur la carte FPGA de la famille Virtex 2 (xc2v6000-6).

Les deux autres tableaux systoliques à niveau de granularité double et quadruple ont des gains de performances spécifiques. Le tableau systolique double (*PE_2×2*) dépasse légèrement le *PE_1×1* en vitesse de traitement, mais permet de comparer des séquences moins longues (6000 nucléotides). Ce système est le plus équilibré au point de vue utilisation de ressources matérielles (mémoire-logique).

Le tableaux systolique quadruple (*PE_4×4*) présente un gain de vitesse de 8% par rapport au système à éléments simples, mais permet de comparer des séquences d'une longueur de 4200 nucléotides. Ce tableau systolique nécessite plus de ressources logiques que de ressources de mémorisation.

Enfin, une vision d'un système reconfigurable est proposée, elle concerne la possibilité de varier le niveau de granularité selon la longueur des séquences comparées.

ABSTRACT

Bioinformatics plays important roles in biological and medical research. Computational biology and biological data organization related to genomes aim for the formulation and implementation of search strategies. The application of this information will help determine the origin of diseases, in pharmacology and in other commercial applications. Bioinformatics focuses on the development of practical tools for data management and analysis.

Moore's Law observes that the number of transistors that can be integrated doubles every 18 months. However, the amount of genomic data at *GenBank*, an annotated collection of all publicly available DNA sequences, is doubling every six months. The growing gap between the amount of information to be analyzed and integration density implies that different implementation strategies must be considered. This requires a massive design effort to choose the best compromise between three criteria: flexibility, programmability and computational density.

The ultimate objective of this project is to provide a performance increase for the implementation of genetic sequence comparison applications. The two areas where performance enhancements are important are processing speed and maximum sequence length. To accelerate processing speed, two parallel methods are adopted. First, a software parallelization is done. It uses approaches based on two parallel programming standards, OpenMP (Open Multi-Processing) and MPI (Message-Passing Interface). Second, a method of hardware parallelization is followed. It involves a design study and the realizations of processing systems. Simplification of dynamic programming score formulas calculation has produced gains in processing speed and a reduction in the hardware cost of implemented systems. These systems have different granularity levels: simple, double and quadruple. A comparative study of their performances was

performed. These processing systems have resulted in an O(n) complexity instead of O(n^2) in the case of a dynamic programming algorithm implemented in software.

Our systolic array based on a simple processing element (*PE_1×1*) presents an acceleration varying between 13% and 142% compared to existing realizations. As an indication, the proposed *PE_1×1×7400* is able to compare 3.54×10^6 sequences having a length of 7400 nucleotides in only one minute. This systolic array has the highest processing speed for the largest size of sequences (7400 nucleotides) that can be implemented on the FPGA Virtex 2 family (xc2v6000-6).

The two other systolic arrays with double and quadruple granularity levels have specific performance gains. The double systolic array (*PE_2×2*) slightly exceeds *PE_1×1* in processing speed, but it allows comparison of shorter sequences (6000 nucleotides). This system is the more balanced with respect to considerations of physical resources use (memory-logic).

The quadruple systolic array (*PE_4×4*) presents a speed gain of 8% compared to the single element system, but it compares sequences with a maximal length of 4200 nucleotides. This systolic array requires more logic resources than memory resources.

Finally, a reconfigurable system vision is suggested. It concerns the possibility of changing the granularity level, depending on the compared sequences length.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ACRONYMES

ADN	Acide DésoxyriboNucléique
ARN	Acide RiboNucléique
ASIC	Application-Specific Integrated Circuit
BLAST	Basic Local Alignment Search Tool
BSP	Board Support Package
CPU	Central processing Unit
DDBJ	DNA Data Bank of Japan
DNA	Deoxyribonucleic acid
DP	Dynamic Programming
DP AM	Dynamic Programming Approximate matching
DRAP	Design et Réalisation d'Applications Parallèles
ED	Energy Department
EMBL	European Molecular Biology Laboratory
FIFO	First In First Out
FPGA	Field-Programmable Gate Array
GA	Global Alignment
GOPS	Giga Operations Per Second
GPP	General Purpose Processor
HGP	Human Genome Project
HPC	High Performance Computing
HW	Hardware

IP	Intellectual Property
ISE	Integrated Software Environment
LA	Local Alignment
LUT	Look-Up Table
MCUPS	Million of Dynamic Programming Cell Updates Per Second
MPI	Message-Passing Interface
NCBI	National Center for Biotechnology Information
NEWCAS	IEEE International Northeast Workshop on Circuits and Systems
N-W	Needleman-Wunsch
NIH	National Institute of Health
nMOS	Negative Metal-Oxide-Semiconductor
OpenMP	Open Multi-Processing
OS	Operating System
PC	Personal Computer
PCI	Peripheral Component Interconnect
PDGF	Platlet-Drived Growth Factor
PE	Processing Element
P-NAC	Princeton Nucleic Acid Comparator
SW	Software
S-W	Smith-Waterman
TS	Tableau Systolique
VHDL	Very high speed integrated circuit Hardware Description Language
VLSI	Very-Large-Scale Integration

LISTE DES ANNEXES

INTRODUCTION

La bioinformatique est un champ de recherche multidisciplinaire où travaillent conjointement biologistes, informaticiens, mathématiciens et physiciens dans le but de résoudre des problèmes scientifiques posés par la biologie. Dans ce qui suit, nous donnons une brève revue de ce domaine de recherche en déterminant notre plan d'action.

Historique et description

L'histoire du calcul biologique remonte aux années 1920 où les scientifiques pensaient déjà à établir des lois biologiques de l'analyse des données seulement par induction (par exemple A.J. Lotka, *Elements of Physical Biology*, 1925). Cependant, le développement des ordinateurs puissants, et la disponibilité des données expérimentales qui peuvent être aisément traitées par calcul, ont lancé la bioinformatique comme discipline indépendante [1].

Le lien entre l'informatique et la biologie est naturel pour plusieurs raisons. En premier lieu, le taux phénoménal auquel des données biologiques sont produites présente des défis, car des quantités massives de données doivent être stockées, analysées, et rendues accessibles. En second lieu, la nature des données est souvent sujette à une analyse, et par conséquent au calcul. Ceci s'applique en particulier à l'information sur les séquences de protéines et à l'organisation temporelle et spatiale de leur expression dans la cellule codée par l'ADN. En troisième lieu, il y a une analogie forte entre une structure d'ADN et une machine à états dans un système informatique (dépendance de données entre les différents états du système complet) [1].

Intérêt scientifique du domaine de comparaison de séquences

Un nouveau gène une fois trouvé, les biologistes n'ont souvent pas une indication sur sa fonctionnalité. Une approche commune pour déduire un nouveau gène séquencé est de trouver la similarité avec d'autres gènes ayant des fonctionnalités connues. Un exemple représentatif d'une telle découverte biologique au moyen de la recherche de similarité est arrivé en 1984. Des scientifiques ont utilisé une simple technique computationnelle pour comparer le nouveau oncogène *v*-sis cancéreux, avec tous les autres gènes connus à l'époque. Avec beaucoup d'étonnement, le gène cancéreux avait une similarité avec un gène normal nécessaire pour la croissance et le développement, appelé PDGF (*platlet-drived growth factor*) [2]. Après avoir découvert cette similarité, les scientifiques ont suspecté le fait que le cancer pourrait être causé par une croissance normale d'un gène transformé, à l'origine, dans le mauvais moment. Ce qui veut dire qu'un bon gène faisait une chose correcte, mais au mauvais moment.

Établir un lien entre les gènes cancéreux et les gènes normaux est parmi les premiers succès du domaine de comparaison de séquences. D'autres applications et algorithmes de comparaison ont suivi. Actuellement, les approches de la bioinformatique sont parmi les techniques dominantes pour la découverte des fonctionnalités génétiques [2]. Dans le premier chapitre, nous détaillons les techniques et caractéristiques des algorithmes basés sur la programmation dynamique, dédiés au domaine de comparaison de séquences, spécialement la qualité de leurs résultats de comparaison renvoyés.

Défi d'explosion des bases de données génétiques

Avec l'arrivée de l'ère du génome, la bioinformatique joue des rôles importants dans la recherche biologique et médicale. La biologie computationnelle et

l'organisation des données biologiques se rejoignent aux objectifs du génome pour appliquer cette information génétique. La bioinformatique se concentre sur le développement d'outils pratiques pour la gestion des données et leur analyse [3].

Durant les dernières années, il y a eu une croissance explosive des données biologiques provenant des projets du génome, de la protéomique (technique permettant l'analyse simultanée de plusieurs protéines) [4]. Beaucoup de données proviennent aussi de l'expansion rapide dans la numérisation des données biologiques des patients. Des techniques informatiques à hautes performances sont nécessaires pour comprendre et analyser l'information biologique codée par les séquences d'ADN, ce qui constitue un défi.

La comparaison des séquences génétiques constitue une étape initiale dans la résolution de plusieurs problèmes en bioinformatique, où la similitude entre les séquences est souvent recherchée [4].

Objectifs du projet

Ce travail vise la conception et la réalisation d'une plateforme qui élabore une parallélisation matérielle du calcul HPC (*high performance computing*), tout en améliorant la vitesse de traitement et en réduisant le coût matériel sur puce. Ceci devrait répondre au défi d'explosion de taille des bases de données génétiques. Plus spécifiquement nous visons les points suivants :

1. Déterminer les gains de performance qu'une parallélisation logicielle peut apporter aux applications de comparaison de séquences;
2. Simplifier la conception d'une unité de traitement élémentaire basée sur la programmation dynamique, dans le but de réaliser une parallélisation matérielle;
3. Concevoir d'autres unités de traitement en variant le niveau de granularité;

4. Déterminer le niveau de granularité optimal (nombre de cellules traitées à chaque itération) du tableau de traitement systolique qui permet une accélération maximale de traitement, avec un coût matériel minimal (meilleur gain de performances).

Contribution

La contribution de ce mémoire se résume essentiellement à l'accélération du traitement parallèle en matériel, pour des applications de comparaison de séquences génétiques. Trois architectures innovatrices sont proposées grâce à l'introduction de deux nouvelles équations qui régissent le comparateur élémentaire de la cellule de programmation dynamique.

La conception de trois éléments de traitement de type simple, double et quadruple, a permis de réaliser trois systèmes à niveaux de granularités correspondants à ces éléments. Le premier système (simple) a permis de traiter des séquences d'une longueur de 7400 nucléotides, avec un gain de vitesse de traitement qui varie entre 13% et 142% par rapport aux systèmes similaires existants. Une réduction d'utilisation de ressources matérielle de 25% est obtenue, comparativement au système présenté par Yu et al. [5].

Les deux autres systèmes, double et quadruple, traitent des séquences moins longues. Ils présentent des gains de vitesse qui varient entre 2% et 8% par rapport au système à élément simple. Ces réalisations ont permis de tirer des conclusions quant aux performances obtenues en fonction des niveaux de granularité associés, et des coûts matériels requis pour leurs implémentations.

Organisation du mémoire

Ce mémoire contient quatre chapitres. Le premier chapitre donne un rappel des concepts de base du domaine de comparaison de séquences. Il contient aussi une définition de la problématique de notre sujet de recherche. Le deuxième chapitre est une revue de littérature, où les principales réalisations matérielles et logicielles sont présentées, avec une analyse critique de ces travaux.

Le troisième chapitre expose nos deux parallélisations logicielle et matérielle. Les approches de parallélisation logicielle sont détaillées avec leurs résultats obtenus. Pour la parallélisation matérielle, une étude de conception est présentée, avec des explications des principes théoriques utilisés. Ce chapitre détaille aussi la composition des trois systèmes de comparaison réalisés, à niveaux de granularité variés.

Le quatrième chapitre résume les résultas des trois systèmes réalisés et détaillés au troisième chapitre. Une étude comparative analyse les apports de performances obtenus.

Le mémoire se termine avec une conclusion générale, ainsi qu'une vision des travaux futurs qui pourrait donner une continuation à ce projet de recherche.

CHAPITRE 1 CONCEPTS DE BASE ET PROBLÉMATIQUE DU DOMAINE DE COMPARAISON DE SÉQUENCES

Dans ce premier chapitre, nous présentons un rappel des concepts de base reliés à la nature de l'information génétiques. Principalement, nous décrivons sa localité, sa forme, son codage, ainsi que son évolution historique. Nous présentons aussi les techniques utilisées pour comparer cette information sous forme de séquences d'ADN ou de protéines. Nous détaillons spécifiquement la méthode de programmation dynamique qui constitue notre objectif d'étude. La dernière section de ce chapitre présente la problématique du sujet abordé dans ce projet de recherche face au défi d'explosion de la taille des bases de données génétiques.

1.1 Cycle de vie

Les organismes vivants étant composés à la base de cellules, l'étude de la vie revient alors à étudier la cellule.

Malgré la grande variété des cellules existantes, elles ont certaines caractéristiques en commun, comme le cycle de vie. Ce cycle commence par la naissance, il comporte le besoin de se nourrir et l'étape de division. Il s'achève par la mort. Les cellules suivent des procédures complexes sous forme de réactions chimiques pour la composition/décomposition de la matière, ainsi que la génération des signaux indiquant le moment de se nourrir ou de mourir. Ces algorithmes complexes qui contrôlent ce cycle de vie restent toujours en dehors de notre compréhension [3].

La cellule peut être vue comme un système mécanique complexe contenant des parties en mouvement. Non seulement est-elle capable de mémoriser l'information dans une copie complète d'elle-même, mais elle dispose aussi de tous les mécanismes nécessaires pour la fabriquer. Cette information est conservée dans trois types de molécules primaires: ADN (acide désoxyribonucléique), ARN (acide ribonucléique), et les protéines. Grosso modo, l'ADN d'une cellule constitue une large bibliothèque qui décrit le fonctionnement de la cellule. L'ARN agit comme transporteur de courtes parties de cette bibliothèque vers différents endroits de la cellule. Ces petites parties d'information sont utilisées pour former les protéines. À leur tour, les protéines forment les enzymes responsables du fonctionnement de la cellule.

1.2 Information génétique

L'ADN est le support de l'hérédité ou de l'information génétique, car il constitue le génome des êtres vivants, et se transmet en totalité ou en partie lors des processus de reproduction. L'ADN détermine la synthèse des protéines, qui est un processus par lequel une cellule assemble une chaîne de protéines. Ceci en combinant des acides aminés isolés présents dans son cytoplasme. La synthèse des protéines se déroule, au moins, en deux étapes: la transcription de l'ADN en ARN messager et la traduction de l'ARN messager en une protéine. Dans les cellules eucaryotes, l'ADN est contenu dans le noyau. Dans les cellules procaryotes, l'ADN est contenu dans le cytoplasme [3].

1.2.1 Composition du nucléotide d'ADN et génome

Un nucléotide est composé de trois parties: un groupement phosphate, un sucre correspondant à un désoxyribose, et une base azotée purique (A ou G), ou pyrimidique (T ou C). Une hélice à doubles brins antiparallèles de nucléotides forme

l'ADN. Ils sont toujours étroitement reliés entre eux par des liaisons hydrogène ('liaisons H' ou 'ponts H') formées entre les bases complémentaires A-T et G-C.

Les nucléotides sont complémentaires entre eux. Ainsi, l'adénine est complémentaire à la thymine et la guanine est complémentaire à la cytosine. Deux liaisons hydrogènes retiennent ensemble la paire A-T et trois retiennent la paire G-C. En conséquence, les deux brins d'ADN sont aussi complémentaires, car les purines (A et G) d'un brin font toujours face à des pyrimidines de l'autre brin (T et C) [6].

La combinaison des trois bases code un acide aminé, ou forme les codants d'arrêt UAG, UGA, UAA. La longue séquence d'ADN liée à des protéines telles que les histones forme "la chromatine" que l'on trouve au niveau du noyau d'une cellule [7].

Un gène est caractérisé par sa longue séquence de nucléotides d'ADN qui constitue le support d'un message codé déterminant la nature des protéines, c.-à-d. le nombre, l'ordre, et l'identité de leurs acides aminés [7] L'ensemble du matériel génétique d'un individu ou d'une espèce est le génome. Les gènes ne constituent qu'une partie du génome. Celui-ci est constitué de molécules d'acides nucléiques : l'ADN et l'ARN. Le génome humain est réparti sur 46 chromosomes [8], dont une copie est présente dans chacune des cellules.

1.2.2 Codage de l'information génétique et relations évolutionnaires

L'information génétique est codée sous forme d'un système de correspondance entre la séquence nucléotidique de l'ARN messager (et donc celle de l'ADN) et la séquence en acides aminés de la protéine. La séquence d'acides nucléiques est une combinaison de 4 nucléotides A, C, T, G. Leur enchaînement codera les 20 acides aminés possibles dans les protéines. Pour qu'il soit fait d'une manière non ambiguë, le codage d'acides aminés doit se faire avec 3 bases, on obtient alors $4\times4\times4=64$ codons. Il est possible donc de coder 63 différents acides aminés et

un codon signifiant 'pas d'acide aminé' (codon d'arrêt). Dans la nature, un acide aminé est codé par plusieurs enchaînements différents de bases : le code est dit dégénéré. C'est pour cette raison qu'on parle de redondance du code génétique. Par exemple, l'acide aminé 'leucine' peut être codé par 6 enchaînements différents : UUA, UUG, CUU, CUC, CUA, CUG [6]. Nous remarquons que la thymine (T) est remplacée par l'uracile (U), car les codons sont portés par l'ARN messager, utilisé par la cellule pour transmettre l'information génétique à l'extérieur du noyau.

Dans une même famille d'acides nucléique ou de protéine, les variations internes à cette famille constituent une grande source d'information de l'évolution biologique. L'information des séquences est dorénavant grandement disponible, ce qui permet de détecter un gène ancêtre, comme un ARN ribosomal, ainsi que quelques protéines. La détection d'ancêtre peut se faire en suivant le chemin arrière dans l'arbre de vie (arborescence de classification des espèces vivantes) vers la racine [3].

La description de l'arbre de vie et le catalogage de la biodiversité, tirent profit des bases de données des projets de comparaison du génome. La description et l'analyse des relations évolutionnaires entre les protéines basées sur l'analyse du génome constitueront une avancée majeure. Spécialement, ceci sert à comprendre l'évolution de la structure du génome et l'organisation biochimique et structurelle des organismes [9]. De plus, les séquences de protéine sont prédites par traduction de séquences d'ADN qui sont des copies des séquences d'ARN messager.

1.3 Projet du génome humain et bases de donnée génétiques

Le Projet du Génome Humain (*HGP*) est un projet entamé officiellement en 1990 par une coordination entre le Département d'Énergie (*ED*) et l'Institut National de Santé (*NIH*) des États-Unis. Le *Wellcome Trust* (Royaume-Uni) est devenu l'un

des principaux partenaires; des contributions supplémentaires venaient du Japon, de France, d'Allemagne, de Chine et d'autres [10].

La mission du projet était d'établir le séquençage complet du génome humain. Le projet était initialement prévu pour une durée de 15 ans, mais l'accélération des progrès technologiques a avancé la date d'achèvement en 2003. Il porte l'ensemble de l'information génétique, distribuée sur 20000-25000 gènes dans l'ADN humain. Entre autre, ce projet avait comme objectifs l'identification des 3 milliards de paires des bases chimiques qui composent l'ADN humain, le stockage des informations dans des bases de données, ainsi que l'amélioration des outils d'analyse de données [10].

La collection des séquences d'ADN dans les bases de données *GenBank* a commencé en 1974 par le groupe de biologie et biophysique théorique au laboratoire national de *Los Alamalos* du Nouveau-Mexique (*USA*). En Europe, le laboratoire de biologie moléculaire européen (*EMBL*) [11]était fondé en 1980. Du coté du Japon, leur base de données *DNA DataBank* (*DDBJ*) [12] a vu le jour en 1984. *GenBank* est actuellement sous la protection du centre national de l'information biologique (*NCBI*) [13]. Par la suite, *GenBank*, *EMBL*, et *DDBJ* ont formé la collaboration de base de données internationale de séquences de nucléotides [9][13]. Cette collaboration prend en charge la facilité d'échange quotidien de données [3]. Des bases de données d'autres types de séquences sont décrites chaque année dans le journal *Nucleic Acids Research* [14].

Dans une base de données, une entrée de séquence comporte un nom de fichier ainsi que des fichiers de séquences d'ADN ou de protéine. Ces fichiers peuvent contenir des informations additionnelles sur la séquence, comme sa fonction, ses mutations, les protéines encodés, ses sites réguliers et ses références [3]. Ces informations sont annotées dans un format déterminé pour qu'elles soient prêtes aux consultations [3]. Cette dernière pourrait se faire via les sites Internet de ces bases de données qui disposent d'interfaces dédiées à cette fin [11][12][13]. L'augmentation de la taille des ces bases de données est continuelle. Par exemple, dans *GenBank* il y

avait 1.26×10^9 bases en décembre 1997, il y en avait 39×10^9 bases en avril 2004 [3]. Une illustration de l'évolution historique des données génétiques est présentée par la Figure 1.7 de la section 1.7.

1.4 Alignement de séquences génétiques

L'alignement de séquence est une manière de disposer les composantes (nucléotides ou acides aminés) des ADNs, des ARNs, ou des séquences primaires de protéines. Le but est d'identifier les zones de concordance qui traduisent des similarités ou dissemblances entre les séquences [3].

Il y a deux types d'alignement de séquences, global et local [3]. Dans l'alignement local (*LA*), des bouts de séquences ayant la plus grande densité de ressemblance sont alignées, ce qui produit une ou plusieurs régions de similarité ou des sous alignements. L'alignement local est plus approprié aux séquences qui ont des régions semblables au long de certaines de leurs longueurs, mais différentes ailleurs. Dans l'alignement global (*GA*), les deux séquences sont alignées sur toute leur longueur. La Figure 1.1 (a) donne un alignement global de deux séquences, alors que la Figure 1.1 (b) donne un alignement local de deux autres séquences d'ADN, où il trouve une région de forte homogénéité.

```
TTAC- TTAC- -ATCG-A-TCGAAA-T
||  |  ||      ||         |       |
TT- CGTTCG- -ATGCCC -CCCCTTT
```

```
--------ATCG -A---------
         ||||  |
--------ATCGGA --------
```

Figure 1.1 (a) Alignement global. (b) Alignement local.

L'alignement de séquences est utilisé pour découvrir l'information évolutionnaire et structurale des séquences génétiques (ADN, protéine). Il est important d'obtenir un alignement optimal (maximum de similarité) pour découvrir cette information. À proprement dit, des séquences sont similaire si elles ont

probablement la même fonctionnalité. Ceci signifie, soit un rôle régulier dans le cas d'ADN, soit une fonction biochimique ou une structure tridimensionnelle similaires dans le cas des protéines. En outre, si deux séquences de deux organismes différents sont similaires, ils pourraient avoir la même séquence ancêtre en commun. De ce fait, les deux premières séquences peuvent être définies comme des séquences homologues. Donc, un alignement indiquera les changements que les deux séquences ont vécu durant leur évolution par rapport à leurs séquences ancêtres [3].

L'alignement de séquences peut être effectué à l'aide de trois méthodes: analyse de matrice de points (*dot-matrix*), algorithmes de programmation dynamique, et la méthode de *k*-uplets de mots [3]. Le principe de ces trois méthodes est décrit brièvement dans ce qui suit.

1.4.1 Analyse par matrice de points

L'approche d'analyse par matrice de points produit implicitement une famille d'alignements pour des régions de séquences. Cette approche est qualitative même si elle consomme un temps considérable sur une grande échelle. Avec cette méthode, ont peut identifier visuellement certaines caractéristiques des séquences telles que les insertions et les suppressions à partir d'un graphique d'une matrice de points.

Pour construire la matrice de point, les deux séquences sont écrites sur la première rangée et la colonne la plus à gauche d'une matrice bidimensionnelle. Un point est placé à toute position de similitude (*i, j*) entres les caractères des deux séquences. Ceci revient à faire un graphique récurent, où l'intensité des points tracés dépend du degré de similitude entre les deux séquences comparées. Le graphique des points étroitement liés apparaîtra comme une seule longue ligne située sur la diagonale principale de la matrice de comparaison.

Le principal avantage de cette méthode est que tous les résidus de similitudes entre les deux séquences sont trouvés. Ceci laissera le choix à l'utilisateur d'identifier

les régions les plus significatives sous forme de ligne de points parallèles à la diagonale (Figure 1.2). Puis, l'alignement peut être effectué en déterminant le résultat maximal des régions de séquences concernées, par l'utilisation de l'une des deux autres méthodes, la méthode de *k*-uplets de mots ou la programmation dynamique. La dernière méthode est automatique et permet de produire un alignement optimal [3].

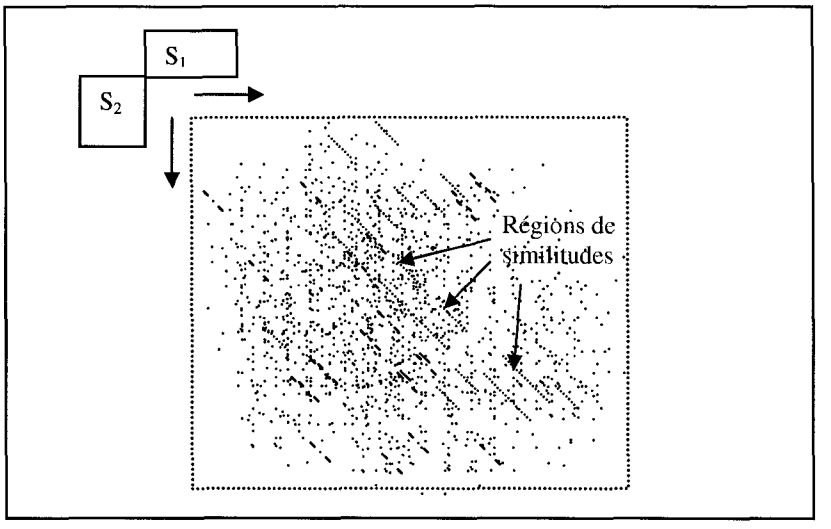


Figure 1.2 Analyse de deux séquences d'ADN par la matrice de points.

1.4.2 Méthode de *k*-uplets de mots

Les méthodes basées sur la recherche de mots, aussi connues sous le nom *k*-uplets de mots, sont des méthodes heuristiques qui ne garantissent pas un résultat d'alignement optimal. Cependant, elles sont plus rapides que les solutions basées sur la programmation dynamique. Ces méthodes sont utilisées dans les recherches de bases de données à grande échelle, où une grande proportion des séquences cibles n'auront essentiellement pas de similitude avec la séquence requête. Les méthodes de *k*-uplets sont, spécialement, utilisées dans les implémentations des outils de recherche de bases de données FASTA [3][15] et la famille d'outils *BLAST* [3][16].

Une méthode de *k*-uplets identifie une série de courtes sous-séquences ('mots') dans la séquence objectif, pour qu'elles soient comparées avec les séquences cibles de la base de données. Les positions relatives des mots dans les deux séquences comparées sont soustraites de manière à obtenir un décalage. Ceci indiquera une région d'alignement si plusieurs mots distincts produisent le même décalage. Si cette région est détectée, ces méthodes appliquent des critères d'alignements plus sensibles. Ainsi, de nombreuses comparaisons avec les séquences ayant une mauvaise similitude seront éliminée [3].

Dans la méthode de *FASTA*, l'utilisateur définit une valeur *k* comme longueur des mots permettant d'interroger la base de données. La méthode est lente mais plus sensible pour des petites valeurs de *k*, qui sont également préférées pour les recherches impliquant de courtes séquences cibles. Les méthodes de recherche de la famille *BLAST* fournissent un certain nombre d'algorithmes optimisés pour certains types de requêtes, telle que la recherche de séquences ayant des relations de similitudes distantes [3]. *BLAST* a été développé pour fournir une solution alternative plus rapide par rapport à *FASTA*, mais avec moins de précision. *BLAST*, comme *FASTA*, utilise une recherche par mot de longueur *k*, mais en évoluant uniquement les similitudes de mots les plus significatifs, pas tous les mots comme le fait *FASTA*. Des implémentations des deux solutions existent sur des portails Web tels que l'*EMBL FASTA* [15] et *BLAST NCBI* [16].

1.4.3 Programmation dynamique

La méthode de la programmation dynamique peut être appliquée pour produire un alignement global via l'algorithme de Needleman-Wunsch [17], et un alignement local via l'algorithme de Smith-Waterman [18].

Typiquement, les alignements de protéines utilisent une matrice de substitution. Cette matrice attribue des résultats de similarité ou de dissimilitude aux acides aminés. Elle fait correspondre une pénalité de trou d'un acide aminé d'une

séquence à un trou dans l'autre séquence. Les alignements d'ADN et d'ARN utilisent une matrice de résultat. Souvent, on attribue un résultat positif pour la similitude, un résultat négatif pour la dissimilitude, ainsi qu'une pénalité de trou négative [2].

La méthode de programmation dynamique constitue notre objectif d'étude, de simplification, et d'implémentation. Nous allons détailler ses deux principaux algorithmes dans la section (1.5).

1.5 Principes de la programmation dynamique

1.5.1 Définition

Inventée par le professeur Richard Bellman (mathématicien, 1920-1984), la programmation dynamique (*DP*) permet de résoudre au moyen d'un ordinateur tout problème d'optimisation dont la fonction objectif se décrit comme la somme de fonctions monotones non-décroissantes des ressources [19]. Concrètement, cela signifie que l'on va pouvoir déduire la solution optimale d'un problème à partir d'une solution optimale d'un sous-problème [2].

Le premier algorithme basé sur la programmation dynamique pour la comparaison des séquences d'ADN fût publié par Saul Needleman et Christian Wunsch [17]. Russel Doolittle et ses collègues utilisaient des heuristiques pour déterminer la similarité entre les gènes cancéreux et le gène *PDGF* [2] en 1983. Quand Needleman et Wunsch ont publié leur papier en 1970, ils ne savaient pas qu'un algorithme très similaire était publié, deux ans avant, dans un article pionnier dans le domaine de reconnaissance automatique de la parole [20]. Plus tôt, Vladimir Levenshtein a introduit la notion de la distance d'édition, mais il n'a pas établi un algorithme pour la calculer [2]. L'algorithme de l'alignement local introduit par Temple Smith et Michael Waterman en 1981 est devenu, rapidement, l'outil d'alignement le plus populaire dans le calcul biologique [2].

1.5.2 Programmation dynamique et alignement de séquences

La programmation dynamique est utilisée pour l'alignement de séquence d'ADN ou de protéines. Cette méthode de calcul est très importante, car dans un sens mathématique elle produit le plus haut résultat, ou l'alignement optimal entre deux séquences [3]. Les deux alignements global et local peuvent être obtenus en effectuant un simple changement dans la formule de la programmation dynamique. Un programme qui produit un alignement global sera basé sur l'algorithme de Needleman-Wunsch (N-W) [17], Alors qu'un programme qui produit un alignement local sera basé sur l'algorithme de Smith-Waterman (S-W) [18].

Les algorithmes basés sur la programmation dynamique sont utilisés pour l'alignement de séquences en utilisant la distance d'édition *(edit distance)* au lieu de la similarité entre les séquences. Nous présentons, ci-dessous, la formule de l'algorithme de N-W et celle de l'algorithme de S-W, ainsi que le type des résultats qu'ils renvoient.

1.5.3 Formules de calcul des résultats

Soient deux séquences génétiques $M=m_1m_2...m_i...m_m$ et $N=n_1n_2...n_i...n_n$. Pour calculer le résultat maximal de similarité entre deux sous séquences $m_1...m_i$ *et* $n_1...n_j$, la formule récursive de Needleman-Wunsch est utilisée [2]:

$$S_{i,j} = \max \begin{cases} S_{i-1,j} - \sigma \\ S_{i,j-1} - \sigma \\ S_{i-1,j-1} - \mu, if : m_i \neq n_i \\ S_{i-1,j-1} + 1, if : m_i = n_i \end{cases} \qquad (1)$$

où σ est la pénalité pour un espace de longueur 1, μ est la pénalité pour aligner les deux caractères à la position i et j, et $S_{i,j}$ est le résultat à cette position. L'équation (1) peut également être exprimée sous la forme suivante [21]:

$$\forall i : S_{i,0} = G_{pénalité} \times i, \forall j : S_{0,j} = G_{pénalité} \times j$$
$$\forall i, j, i \times j \neq 0 :$$
$$S_{i,j} = \max \begin{cases} S_{i-1,j-1} - G_{pénalité} \\ S_{i,j-1} - G_{pénalité} \\ S_{i-1,j} + D(m_i, n_j) \end{cases} \quad (2)$$

où $G_{pénalité}$ est la pénalité de trou et $D(m_i , n_j)$ est la distance entre les caractères m_i et n_j.

Un résultat biologique ayant une signification particulière est donné par le système d'équations (3). Ce résultat permet de calculer le nombre minimum d'insertion et de suppression entre deux séquences [21].

$$\begin{cases} G_{pénalité} = -1 \\ \forall i, D(m_i, m_i) = 0 \\ \forall i, j, i \neq j, D(m_i, n_j) = -2 \end{cases} \quad (3)$$

Le calcul de la distance d'édition permet de représenter le nombre minimal d'opérations nécessaires pour transformer une séquence *S1* vers une autre séquence *S2*. Par exemple, pour transformer la séquence AGCTATA vers la séquence TACCGTA, deux opérations de suppression, deux opérations d'insertion, et une opération de substitution sont nécessaires selon l'ordre mentionné dans la Figure 1.3.

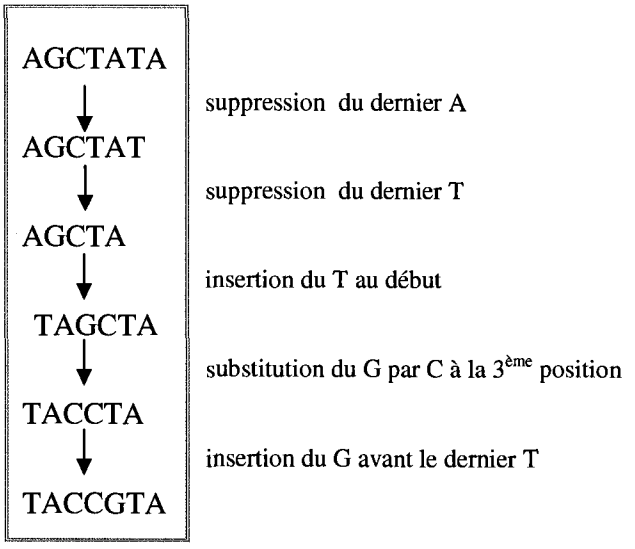


Figure 1.3 Opérations de la distance d'édition pour transformer AGCTATA vers TACCGTA.

L'équation (3) calcule ce nombre en effectuant la somme des opérations d'insertion et de suppression (*deletion*) entre ces deux séquences, ainsi que la substitution qui revient à faire une suppression et une insertion (Figure 1.4.a). Ceci permet de déduire un alignement global (Figure 1.4.b).

D(*i,j*)	N(*j*)	T	T	A	C
M(*i*)	0	1	2	3	4
T	1		1	2	3
T	2	1			2
C	3	2	1	2	
G	4	3	2	3	

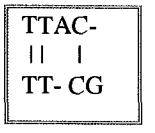


Figure 1.4 (a) Matrice de calcul de la distance d'édition. (b) Alignement global.

Pour un résultat d, nécessitant les valeurs des voisins : diagonal a, vertical b, et horizontal c, la formule (2) prendrait la forme suivante [22]:

$$d = \min \begin{cases} b + 1 \\ c + 1 \\ \begin{cases} a \rightarrow if\ (m_i = n_j) \\ a + 2 \rightarrow if\ (m_i \neq n_j) \end{cases} \end{cases} \qquad (4)$$

Le calcul de la distance d'édition a une propriété très intéressante, découverte par Lipton et Lopresti [22]. Les valeurs des voisins horizontal et vertical (b et c) dans la matrice des résultats diffèrent de ± 1 par rapport aux voisins diagonaux ($a \pm 1$, $d \pm 1$). La Figure 1.5, ci-dessous, illustre une présentation générale de ces valeurs dans la matrice des résultats.

D (i, j)	**N(j)**	..	n_j	..
M(i)		..	j	..
..	..	a	$b=$ $a\pm1$, d±1	
$\boldsymbol{m_i}$	i	$c=$ $a\pm1$, d±1	d	
..	..			⋱

Figure 1.5 Présentation générale des valeurs du voisinage de d dans la matrice des résultats.

Ainsi Lipton et Lopresti ont simplifié le calcul du résultat d dans toute case de la matrice des résultats, donné par la formule suivante [22]:

$$d = \begin{cases} a & if\ (b=a-1)\ \ AND\ \ (c=a-1)\ \ OR\ \ (m_i = n_j) \\ a+2 & if\ (b=a+1)\ \ AND\ \ (c=a+1)\ \ AND\ \ (m_i \neq n_j) \end{cases} \qquad (5)$$

Plusieurs articles présentent des réalisations matérielles du calcul de la distance d'édition [5][23][24][25][26][27]. Les auteurs discutent l'algorithme de S-W, tandis que cet algorithme est employé pour trouver l'alignement local. Dans ce cas-ci, l'algorithme de S-W garantie de trouver le meilleur alignement de segments de séquences objectif et cible ayant un maximum de similitude [2]. Son prédécesseur, l'algorithme de N-W, donne un résultat final dans la dernière cellule de la matrice des résultats. Un alignement global est obtenu en faisant un chemin arrière (*trace back*).

Les deux algorithmes emploient des formules de calcul de résultats différentes, et produisent des résultats différents. L'algorithme de S-W inclut des résultats négatifs pour des dissimilitudes de caractères. Quand les résultats de la programmation dynamique deviennent négatifs, ils sont réinitialisés à zéro. Ceci permet de mettre fin à l'alignement en cours, et de commencer un nouvel alignement local. Le résultat maximal permettra de déterminer le début de la zone ayant un alignement local optimal entre les deux séquences comparées. Ainsi, pour deux séquences M et N, la formule de l'algorithme de Smith-Waterman est la suivante :

$$S_{i,j} = \max \begin{cases} S_{i-1,j-1} + sub(N(i), M(j)) \\ S_{i,j-1} + del(N(i)) \\ S_{i-1,j} + ins(M(j)) \\ 0 \end{cases} \qquad (6)$$

où $S_{i,j}$ est le résultat à la position *(i, j)* des deux séquences N et M et $sub(N(i), M(j))$ est le résultat pour aligner les caractères à la position *(i,j)*. Finalement, $del(N(i))$ et $ins(M(j))$ sont, successivement, les pénalités de

suppression et d'insertion dans les séquences *M* et *N*, prenant des valeurs négatives [3].

Avec cette manière, l'application de la formule (6) permet de remplacer toutes les valeurs négatives des résultats par des zéros dans la première ligne et la première colonne, ainsi que pour toutes les cases de la matrice de calcul des résultats. La Figure 1.6 (a) donne un exemple de calcul des résultats utilisant la formule de S-W. La Figure 1.6 (b) montre un alignement local de deux sous séquences, faisant parties des deux séquences comparées.

D(*i,j*)	N(*j*)	T	T	G	G	A	T	T
M(*i*)	0	-1	-2	-3	-4	-5	-6	-7
A	-1	0	0	0	0	0	0	0
C	-2	0	0	0	0	0	0	0
G	-3	0	0	2	1	0	0	0
G	-4	0	0	2	3	2	2	1
T	-5	0	2	1	3	3	4	4
C	-6	0	1	1	2	2	4	4
A	-7	0	0	0	1	4	3	3

```
GG- T
|||
GGAT
```

Figure 1.6. (a) Matrice des résultats de Smith-Waterman. (b) Alignement local.

1.6 Accélération matérielle de traitement

Le domaine de la comparaison des séquences génétiques ne diffère pas du domaine de comparaison des chaînes de caractères standard, à l'exception de la longueur des séquences comparées. Ceci limite l'utilisation des algorithmes dédiés à ce genre d'applications à cause de la gigantesque longueur des séquences génétiques, où des centaines de millions de caractères doivent être traités pour détecter les régions de similitude. Cette grandeur et la dépendance de données excluent les processeurs à

usage général de cette tâche consommatrice de beaucoup de ressources (unités de calcul, unités de mémorisation) et de temps.

Pour remédier à cette situation, la parallélisation de traitement a été largement investie sous différentes formes [21]. Principalement, des systèmes qui implémentent des parallélisations à gros grain comme les supercalculateurs, les grappes de processeurs ou les réseaux de calcul. Les accélérateurs FPGA sont les plus adaptés pour l'accélération matérielle à fin grain grâce à la bonne performance et la grande flexibilité de cette technologie [4]. Autres technologies sont exploitées sous forme de coprocesseurs, comme les ASICs.

L'analyse à hautes performances des séquences se fonde souvent sur la parallélisation des algorithmes de complexité $O(n^2)$ de la programmation dynamique. L'implémentation de ces algorithmes présente toujours un défi de conception système pour améliorer le calcul biologique à haute performance (HPC) en utilisant différentes approches d'analyse [4].

1.7 Problématique du sujet

La loi de Moore montre que le nombre de transistors intégrables double tous les 18 mois [28]. Cependant, la quantité de données génomique dans *GenBank* [29], une collection annotée de séquences d'ADN disponibles publiquement, double tous les six mois (Figure 1.7). L'espace croissant entre la quantité de l'information à analyser et la densité d'intégration implique la considération de différentes stratégies d'implémentation. Ceci exige un grand effort de conception pour choisir le meilleur compromis entre trois critères : flexibilité, programmabilité et densité computationnelle du système de traitement à réaliser [4].

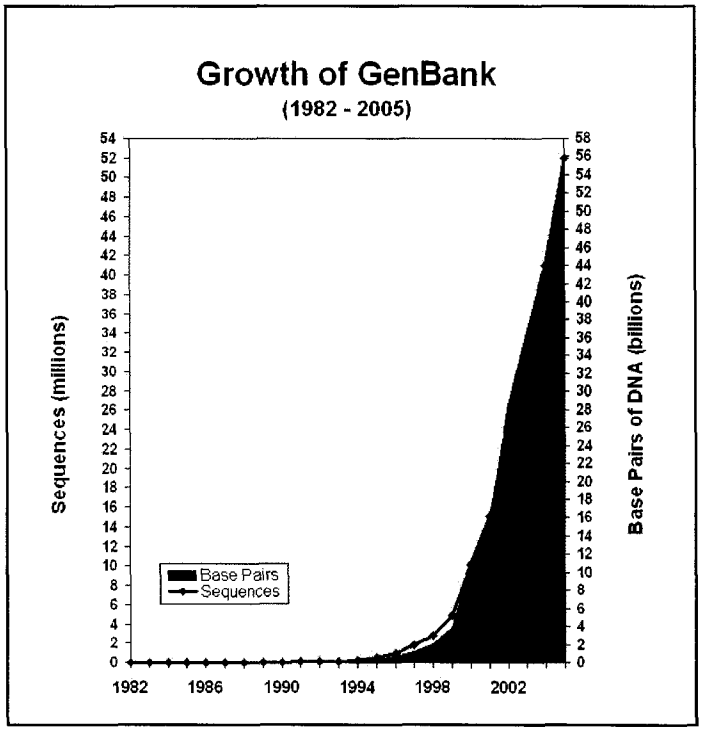


Figure 1.7 Explosion des données génétiques dans *GenBank* [29].

Un des principaux buts des concepteurs des systèmes de calculs biologiques à hautes performances, est de développer une plateforme pour l'analyse efficace des séquences biologiques [21]. Un résultat de comparaison optimal est produit en utilisant les algorithmes de programmation dynamique [2]. Cependant, ces algorithmes ont une complexité $O(n^2)$ et leur implémentation exige une haute parallélisation de calcul.

Tel que mentionné précédemment, les processeurs à usage général (GPP) ne peuvent pas relever efficacement ces défis par eux-mêmes [2]. Au cours des dernières années, l'augmentation de la fréquence des GPPs a ralenti, mais la densité d'intégration des transistors ne cesse d'augmenter [30]. Les nouvelles évolutions technologiques accentuent l'importance de décharger les GPPs des tâches de calculs

intensifs, comme ceux liés aux bases de données biologiques. Ceci crée deux raisons expliquant partiellement la tendance vers l'utilisation des accélérateurs de traitement.

La première raison se résume par le fait que l'accélérateur peut être couplé et intégré sur puce avec une unité centrale de traitement (CPU), afin d'avoir les mêmes avances technologiques. Particulièrement, ces avances concernent une consommation à basse puissance et un meilleur déploiement de l'espace sur puce.

Le deuxième et principal avantage est que la progression du niveau d'intégration implique une grande densité d'éléments de traitement (PEs) et une fréquence opérationnelle effective combinée très élevée.

L'augmentation de la vitesse d'exécution des fonctionnalités spécifiques en utilisant des accélérateurs n'est pas nouvelle, mais les défis spécifiques du calcul HPC en bioinformatique exigent de nouvelles solutions. Le potentiel des FPGAs en tant qu'accélérateurs de calcul reconfigurable résulte du parallélisme à fin grain exigé pour chaque accroissement de la performance du calcul [31]. Cette efficacité supplémentaire produirait un rendement de calcul plus élevé.

Pour le calcul reconfigurable, un effort considérable de définition de la méthodologie de conception reste à faire, car les niveaux moyens de conception sont absents entre les structures logiques et les niveaux d'intégration à moyenne échelle, et la couche application [31]. Il n'y a pas encore l'équivalent du BSP pour un calculateur reconfigurable FPGA qui isolerait le système d'exploitation (OS) des particularités de chaque configuration matérielle spécifique. Il serait utile de soutenir le développement de n'importe quelle future couche OS qui pourrait fonctionner sur un processeur reconfigurable FPGA [31].

Conclusion

Ce chapitre a présenté un survol des connaissances de bases décrivant l'information génétique. Cette information est sauvegardée dans des bases de donnés

génétiques sujettes à des consultations pour des buts de comparaison. Les éléments de la problématique du sujet ont été détaillés. Particulièrement, un effort de conception qui produit une amélioration des outils de traitement pour améliorer la qualité des résultats ainsi que l'efficacité des méthodes de comparaison. Ceci permettra de répondre au défi d'explosion de taille des bases de données génétiques.

CHAPITRE 2 REVUE DE LITTÉRATURE

Nous présentons dans ce chapitre une revue de littérature du domaine de calculs à hautes performances, où nous exposons les types et techniques de parallélisation utilisés ainsi que les systèmes réalisés. Nous discutons aussi les performances atteintes par ces systèmes. Nous justifions par la suite notre choix de la technologie du support matériel cible de notre réalisation.

2.1 Parallélisme logiciel/matériel et transformations

Le parallélisme peut se faire en logiciel ou en matériel. En logiciel, la parallélisation est réalisée au niveau du code avec différentes techniques et stratégies. Il existe une multitude de transformations manipulant les boucles de code haut niveau, pour des buts d'optimisation dans le cas des processeurs superscalaires et vectoriels. Ces transformations maximisent le parallélisme et la localité (spatiale et temporelle) de la mémoire. Elles considèrent les propriétés des tableaux en utilisant l'analyse de dépendance des boucles.

D. Bacon, S. Graham, et O. Sharp [32] ont fait une bonne couverture des techniques traitant des transformations de boucles. On pourrait en citer quelques unes.

La fusion de boucle est une transformation qui permet de regrouper plusieurs boucles en une seule. L'augmentation de la localité temporelle est obtenue par rapprochement de plusieurs instructions manipulant un tableau dans le même corps de boucle [32].

La distribution de boucles est la transformation inverse de la fusion. Dans ce cas, une boucle est séparée en plusieurs boucles dans le but de diminuer le nombre d'instructions et de transferts présents dans chacune d'elles [33].

Le pavage de boucle est une transformation qui permet de diviser l'espace d'itération d'une boucle en blocs. Les nouveaux petits espaces d'itération constituent des nids de boucles, ce qui permet de manipuler moins de données. Cette transformation augmente la profondeur des nids. Ainsi, les possibilités de réutilisation des valeurs à l'intérieur d'un bloc sont améliorées car ceci diminue significativement les conflits de capacités des caches et des bancs de registres. Ceci réduit la quantité des données manipulées par les itérations [33].

L'échange de boucles permet de changer la position de deux boucles dans un nid de boucle. Généralement, une des boucles les plus externes est placée vers une position la plus interne [34]. Cette transformation peut améliorer efficacement les performances de traitement [32].

Le décalage d'instruction dans une boucle est connu aussi sous le nom de pipelinage logiciel (*software pipelining*) ou *retiming* [35]. Cette transformation consiste à définir une date d'ordonnancement multidimensionnel (dans le cas des nids de boucles). La date d'ordonnancement est déterminée en nombre d'itérations [33].

Les transformations citées plus haut conduisent à des améliorations individuelles de la localité d'un programme. La combinaison de plusieurs transformations peut être adoptée, comme la méthode proposée par Carr, McKinley et Tseng [36]. Cette méthode considère les combinaisons possibles de transformations utilisant la fusion, l'inversion, la permutation et la distribution [33]. Les nids de boucle sont optimisés séparément puis combinés avec les nids des boucles adjacentes [33].

D'autres transformations sont spécifiques aux machines parallèles [32]. Bacon et al. mentionnent la distribution des données sur une architecture multiprocesseurs donnée, en utilisant différentes techniques : décomposition en tableaux réguliers, décomposition automatique et alignement, privatisation scalaire et alignement de cache [32]. Les auteurs citent aussi des techniques de parallélisme à gros grains pour

le partitionnement de calcul. Ils citent aussi des techniques d'optimisation des communications (vectorisation des messages, communication collective, pipelinage de messages, élimination des communications redondantes, etc.).

La bibliothèque standard *open source* MPI (*Message - Passing Interface)* est utilisée par Darling et al. dans le cas du parallélisme logiciel [37]. En général, l'utilisation de cette bibliothèque fait appel à une ou plusieurs techniques citées ci-dessus. Dans les travaux que nous avons consultés, il n'y avait pas de détails sur le type de la technique ou la transformée utilisée pour la parallélisation du code.

L'autre catégorie est le parallélisme matériel. Nous commençons par une classification des systèmes existants du calcul à haute performance.

2.2 Classification des systèmes HPC

Comme nous l'avant mentionné à la section 1.7, le fossé croissant entre la quantité de l'information à analyser et la densité d'intégration implique que différentes stratégies d'implémentation doivent être considérées. Ceci exige un grand effort de conception pour choisir le meilleur compromis entre trois caractéristiques : flexibilité, programmabilité et quantité de calcul (GOPS) [4]. La flexibilité implique la possibilité de réutiliser le matériel pour différentes applications. La programmabilité est un attribut au niveau système concernant sa capacité de supporter la reconfiguration au besoin. La densité computationnelle est un attribut *HW/SW* se rapportant à la quantité de calcul par espace, volume, ou puissance. Elle peut correspondre inversement au prix par unité de performance.

En prenant en considération ces trois critères, et en analysant les différents systèmes existants, quatre types de systèmes de traitement sont employés pour l'alignement des séquences génétiques [38]:

Les supercalculateurs à usage général sont les plus flexibles pour l'analyse rapide des séquences, mais ils sont très onéreux.

Les processeurs à application spécifique peuvent réaliser la performance la plus élevée pour un seul algorithme, mais avec des prix allant jusqu'à plusieurs dizaines de milliers de dollars.

Les coprocesseurs programmables visent à obtenir la flexibilité algorithmique des systèmes reconfigurable, et la vitesse et la densité des systèmes à application spécifique. Le coût et la facilité de programmation se situe, généralement, entre les deux autres types. Les accélérateurs de la technologie ASIC ont l'inconvénient du bas volume de production, et donc les coûts de conception ne peuvent pas être répartis sur un grand nombre d'unités. Des exemples de ce type d'accélérateurs sont Kestrel [39] et le GeneMatcher 2 [40].

Le matériel reconfigurable inclut les systèmes basés sur la technologie FPGAs. Ces systèmes ont tendance à avoir une densité d'éléments de traitement inférieure à celle des processeurs à application spécifique, mais ils peuvent être plus rapides que les supercalculateurs. Les FPGAs et la technologie des compilateurs s'améliorent, et les accélérateurs FPGA sont beaucoup moins chers à concevoir. Un exemple d'un accélérateur FPGA récent est le DeCypher de propriété industrielle [41], sur lequel peu de détails de conception ont été publiés.

Dans la section suivante nous présentons une classification des plateformes d'accélération des algorithmes de comparaison génétique. Nous regroupons d'abord les plateformes qui implémentent les algorithmes de la programmation dynamique (ceux de Smith-Waterman et Needleman-Wunsch) sous forme d'accélérateurs FPGA et ASIC. Ensuite, nous présentons quelques plateformes qui implémentent des approches heuristiques, comme BLAST. Ces approches nécessitent moins de puissance de calcul mais elles sont moins précises (au point de vue résultats renvoyés) par rapport aux deux algorithmes standards (S-W et N-W). Aussi nous mentionnons l'approche de parallélisation qui utilise des grappes de processeurs ou

de PCs, comme *MPI*, pour l'accélération qui se base sur les deux approches algorithmique et heuristique.

Nous accordons plus d'intérêts aux accélérateurs implémentant les algorithmes de la programmation dynamique, vu la qualité des résultats de cette approche de traitement. Nous présentons les solutions proposées par les auteurs pour pallier au problème de la force de calcul nécessaire pour ces algorithmes à complexité $O(n^2)$.

2.3 Les accélérateurs FPGA, ASIC, et autres

2.3.1 Le tout premier accélérateur

Les algorithmes de programmation dynamique ont été accélérés en premier par le tableau systolique P-NAC (*Princeton Nucleic Acid Comparator*), une solution ASIC qui se base sur le codage par modulo, proposé par Lopresti en 1987 [42]. Ce comparateur d'acide nucléique est un réseau systolique linéaire qui compare des séquences d'ADN. L'architecture est une réalisation parallèle d'un algorithme de programmation dynamique standard. Les synchronisations de référence d'une implémentation VLSI confirment que, pour son application dédiée, P-NAC est à 200x plus rapide que les mini-ordinateurs courants au moment de sa réalisation. P-NAC est une implémentation *nMOS*, où chaque puce contient 30 éléments de traitement implémentés sur trois rangées de 10 processeurs. Une option de déviation raccourcit le chemin de communication à travers une puce à 10 processeurs.

2.3.2 Les accélérateurs FPGA

Splash et Splash 2 [26] [43] , étaient les premières implémentations FPGA de tableaux systoliques au début des années 90 [21].

Splash est proposé par M. Gokhale et al. [26]. C'est un tableau systolique de logique programmable, il établit le lien entre un tableau systolique à fonction fixe VLSI et un tableau systolique programmable plus versatile. Splash a reçu une mention honorable du prix de Gordon Bell pour synchronisations, en 1989, sur une application qui a comparé une séquence d'ADN avec une bibliothèque de séquences trouvant la plus proche similitude. Le tableau systolique se compose d'une multitude d'éléments (étages) reliées dans une rangée unidimensionnelle. Chaque étage a trois composantes : une carte FPGA de Xilinx, une mémoire locale, et une puce à virgule flottante. Dans l'implémentation de Splash, un élément de traitement se compose de deux modules : un comparateur de caractères et une machine à états finis.

Proposé par D. T. Hoang, Splash 2 est une deuxième génération de tableaux logiques programmables pour du calcul dédié [43]. Le système Splash 2 est un interfaçage constitué de deux cartes FPGA 4010, et 17 cartes logiques programmables. Chaque carte contient 17 FPGA 4010 de Xilinx. Les seize FPGA sont connectées dans un tableau systolique. Le 17ème FPGA est utilisé pour le contrôle des autres cartes. Chaque FPGA est connecté aussi à une RAM statique de 256k x 16. Le langage VHDL est utilisé pour la conception de cette plateforme.

Hoang a implémenté deux tableaux systoliques, l'un est unidirectionnel, l'autre est bidirectionnel. Les simulations montrent que les implémentations réalisées sur la plateforme Splash 2 peuvent traiter une base de données avec un débit de 12 millions de caractère par seconde. Les performances sont meilleures avec un facteur de plusieurs centaines par rapport à une implémentation d'un algorithme de programmation dynamique sur un ordinateur conventionnel [43].

Plus récemment, d'autres implémentations ont été proposées par l'équipe de HokieGene [24], Yu et al. [5], et Bojanić et al. [44].

Pour le système HokieGene, Puttegowda et al. implémentent l'algorithme de S-W, mais le système effectue plutôt l'alignement global. Les auteurs déclarent une

augmentation de performance de 100x, tout en réduisant la complexité matérielle. Leur élément de traitement est implémenté dans quatre slices Virtex I, ou un seul CLB dans Virtex-II.

Le système HokieGene utilise les concepts de reconfiguration au moment de l'exécution (*run-time*). Une carte PCI Osiris à 64-bit est utilisée avec une fréquence de 66 MHz, un serveur Dell *power edge* biprocesseur est utilisé pour loger la carte FPGA. Chaque dispositif de Xilinx XC2V6000 sur le circuit d'Osiris peut supporter 7000 éléments de traitement avec la logique câblée requise pour communiquer et contrôler d'autres parties sur l'appareil. L'élément de traitement fonctionne à 180 MHz sur le dispositif XCV6000 de la catégorie de vitesse -4. Ceci est considéré pour maximiser le rendement du bus PCI de l'architecture développée.

Le tableau systolique de Yu et al. [5] implémente l'algorithme de S-W pour calculer la distance d'édition entre deux séquences comparées. Un élément de traitement dans cette implémentation se base sur la simplification de Lopresti [22] pour la formule de calcul du résultat de la programmation dynamique. Les auteurs codent chaque variable de cette formule avec un seul bit, ce qui leurs a permis d'avoir une implémentation de l'élément de traitement sur 4 slices du Virtex de Xilinx, où 8 registres sont utilisés pour mémoriser les données, 4 LUTs sont nécessaires pour l'implémentation de la logique combinatoire. Un total de 4032 éléments de traitement sont implémentés sur une carte Virtex I XCV1000E-6 avec une fréquence de travail de 202 MHz. Cette réalisation ne nécessite pas une reconfiguration au moment de l'exécution (*run-time reconfiguration*).

Le circuit à haute vitesse de Bojanić et al. [44] utilise aussi la simplification de Lipton et Lopresti [22]. Ils utilisent aussi un codage des variables de la formule du résultat de la programmation dynamique sur 1 bit uniquement. Dans cette conception, le tableau systolique est composé par un groupage de 4 éléments de comparaison, appelé PE^4. Le PE^4 permet de comparer deux caractères des deux séquences d'ADN à

la fois. Ceci lui permet d'avancer un rendement double par rapport à d'autres applications. Ce système ne nécessite pas une reconfiguration du FPGA.

Bien que le langage VHDL soit utilisé dans la majorité des implémentations, Goccione et Keller ont utilisé *Jbits* pour avoir un temps de compilation plus rapide et un bon temps d'exécution d'une reconfigurabilité partielle [23]. Dans cet article, les auteurs implémentent l'algorithme de S-W, et ils utilisent aussi les simplifications de Lipton et Lopresti [22] pour leurs considérations logiques. Par contre, la conception se distingue par l'utilisation du principe de la reconfigurabilité au moment de l'exécution sur une seule carte FPGA. Le système peut comparer plus de trois millions de caractères par seconde.

Un autre système reconfigurable est celui de Jacobi et al. [45]. Dans cet article, les auteurs présentent un prototype d'architecture systolique reconfigurable dédiée au traitement des problèmes solubles par des algorithmes de programmation dynamique généraux. L'architecture a été modélisée par une méthodologie qui exploite une exécution efficace de l'espace/temps de la transformée de Fourier rapide [45]. Les auteurs ont visé l'alignement de séquences en produisant la similitude entre les séquences (sous-séquences alignées), alors que la plupart des solutions matérielles existantes sont limitées à la comparaison de séquences (résultat final de comparaison). Dans leur système implémenté, la fonction du coût respecte le système de graduation biologique (valeurs des pénalités, négatives, passages à zéro pour l'algorithme de S-W). Leur tableau systolique produit des résultats de comparaison. Pour chaque résultat, un pointeur de trois bits indique sa provenance (de quel prédécesseur dans la matrice des résultats il résulte). Cette information est employée par le serveur hôte, où le tableau systolique implémenté sur FPGA est relié, afin de récupérer les alignements de la matrice de similitude. Pour leur implémentation, les auteurs ont effectué un partitionnement de séquences par logiciel, vue la grande taille des séquences biologiques, mais ils ne donnent pas de détails sur la manière avec

laquelle ils partitionnent les séquences génétique sans perdre de l'information biologique (dépendance de données).

Une autre implémentation FPGA a été proposée par Yamaguchi et al. [46]. Les auteurs présentent une manière de réaliser la recherche d'homologie à grande vitesse avec une carte PCI et un tableau FPGA, reliée à un ordinateur Pentium. À l'aide de cette carte, le temps nécessaire pour comparer une séquence objectif de 2048 éléments à une séquence d'une base de données de 64 millions d'éléments par l'algorithme de S-W est d'environ 34 secondes.

Weaver et al. ont implémenté l'algorithme de S-W en utilisant l'algorithme de resynchronisation (*retiming*) de Leiserson [47] après la procédure de placement dans la carte FPGA, pour augmenter la fréquence d'horloge. Cette technique a permis de doubler la vitesse de traitement par rapport à une implémentation ordinaire [48].

Pour accélérer l'algorithme de S-W, Dydel a proposé un pipelinage de trois niveaux [49]. Une implémentation massivement parallèle est effectuée sur une carte FPGA à deux niveaux. D'abord, une parallélisation interne est effectuée, en faisant beaucoup de copies (jusqu'à 88) de la même unité de traitement qui exécute l'algorithme, dans une puce. Ceci permet de comparer une séquence objectif avec beaucoup d'autres séquences cibles d'une base de données de protéine. Le deuxième niveau proposé est la solution hybride : une grappe d'ordinateurs de bureau, chacun équipé par un accélérateur FPGA sous forme de carte PCI. L'analyse de synchronisation sur l'exécution a indiqué que la fréquence de base est à environ 180 MHz (vitesse du dispositif est à -5). Le programme a été vérifié par une implémentation limitée due à l'architecture Spartan II. Les limitations s'appliquent à la longueur de séquence de 512 résidus de protéine. L'implémentation sur une carte FPGA XC2VP70 s'avère être au moins 200 fois plus rapide que celle sur un Pentium IV 1.7 GHz.

Le Multi-FPGA Configurable de Nallatech est un système récent (2005) composé d'un réseau de plateformes de trois cartes FPGA. Il comporte une carte mère de BenNUEY avec une carte FPGA Virtex-II XC2V3000-4 et une carte secondaire attachée au BenBLUE-II avec deux cartes FPGAs Virtex-II XC2V6000-4 [25]. Dans ce système, l'algorithme de S-W est mis en application fournissant une amélioration de la vitesse de calcul de 200x [25]. Les séquences objectifs sont codées en matériel dans le FPGAs tandis que les différents fichiers des séquences cibles de la base de données sont chargés dans la mémoire centrale. Les séquences de la base de données sont d'abord écrites à travers le réseau de DIMEtalk d'une FIFO en lecture de 16K des FPGAs individuels. Un seul dispositif XC2V6000 est capable de faire une mise à jour de 1260 milliards (1.26×10^{12}) de cellules par seconde. À la fin du traitement, la distance d'édition finale est écrite dans une FIFO en écriture.

Van Court et Herbordt ont proposé une implémentation très versatile avec des composants interchangeables permettant de choisir un type d'alignement (global/local), avec un calcul du résultat ainsi qu'une procédure qui effectue le chemin arrière (*trace-back*) [50]. Les auteurs remarquent que plusieurs anciennes implémentations ont atteint des accélérations impressionnantes, mais pour des applications très spécialisées, traitant une ou quelques options parmi plusieurs. Le problème souligné dans cet article, est la possibilité de décrire une architecture qui implémente une vaste gamme d'options comportementales sans perdre de l'ampleur de l'efficacité. Les auteurs ont réalisé un ensemble de trois types de composants qui constituent les éléments des trois différentes parties du problème d'alignement de séquences en utilisant la programmation dynamique. Pour chaque type de composant ils ont fait de multiples implémentations interchangeables. Cette multitude a permis de choisir, pour chaque application, l'ensemble de caractéristiques requises pour la réalisation de la fonctionnalité. Ceci a permis aussi une utilisation maximale des ressources du FPGA. Leurs analyses estiment une amélioration de 4:1 dans les performances temps-espace.

Les auteurs soulignent un problème de développement des algorithmes en matériel, du fait qu'il y a toute une famille de ce genre d'algorithmes. Les auteurs constatent qu'il y a un grand fossé entre ce que les biologistes ont besoin actuellement et ce que les concepteurs de matériel à applications spécifiques ont produit. Ils soulignent aussi le fait que les utilisations de la recherche approchée par programmation dynamique (*DP AM*) changent considérablement dans leurs ensembles d'entrée, relations de récurrence, résultats de sortie, et ainsi de suite. Cependant, une réalisation matérielle typique implémente juste un ensemble de paramètres et de variations comportementales, souvent sans énoncer quelles hypothèses et variations ont été choisies. Ceci ne satisfait pas les besoins des nombreux utilisateurs potentiels, ce qui limite l'applicabilité de la réalisation.

Pour la recherche approximative, les auteurs séparent cette opération en trois sous opérations, ou trois composants: règle de caractère (bas niveau), cellule de comparaison (relation de récurrence), et un séquenceur (plus haut niveau) [50]. L'implémentation du cœur de la *DP AM* s'est basée sur une logique qui consiste à encapsuler les différences entres les trois types de chacun des composants précédents. Si un certain type est changé, l'implémentation est réalisée avec une manière qui n'affecte pas les autres composants du système,

2.3.3 Les accélérateurs ASIC et VLSI

Pour les technologies ASIC et VLSI, la première accélération de l'algorithme de S-W est BISP proposé par Chow en 1991 [51], suivi par Samba en 1997 [52], Swasad en 2002 [53] et Kestrel en 2005 [39].

Le BISP (*Biological Information Signal Processing*) est un système de comparaison de séquences à grande vitesse, conçu pour supporter les nécessités computationnelles du séquençage génétique. Le cœur du système BISP est un processeur versatile VLSI sur puce qui peut exécuter les fonctions de comparaison de longues séquences. Il produit aussi des alignements globaux et locaux entre deux

d'ADN ou séquences de protéine. Une architecture programmable de tableau systolique a été développée par les auteurs. Le système BISP inclut un grand nombre d'éléments de traitement. Le système initial de BISP se compose de 768 éléments BISP, capables de fournir 6.25 x 10^9 opérations entières par seconde.

SAMBA (*Systolic Accelerator for Molecular Biological Applications*) est un accélérateur matériel de 128 processeurs pour accélérer le procédé de comparaison de séquences. Il fournit une relance au PC ou à la performance d'un poste de travail via une carte peu coûteuse [52]. Le système SAMBA appartient à la catégorie ASIC car le cœur du système est un tableau de processeurs dédiés VLSI, mais le système complet contient une interface mémoire FPGA. Le tableau est relié au poste de travail hôte par une carte mémoire FPGA qui agit en tant que contrôleur du tableau et un mécanisme à grande vitesse qui alimente correctement le tableau et filtre les résultats produits. Le prototype de SAMBA se compose de 32 puces. Chaque puce loge un processeur, menant à un tableau de 128 processeurs. La puce a été conçue à l'IRISA (France) et fournit une puissance computationnelle de 400 millions opérations par seconde. Par conséquent, le tableau en entier peut atteindre une exécution maximale de 12.8 milliards d'opérations par seconde.

Le processeur parallèle Kestrel de l'UCSC est un coprocesseur sur une seule carte (*single-board*) avec un tableau linéaire de 512 éléments de traitement SIMD (*single-instruction, multiple-data*) 8 bits [39]. Le système a été conçu et réalisé à l'université de Santa Cruz, en Californie, entre 1993 et 1995, où des applications en bioinformatique ont motivé le développement d'un appareil d'analyse de séquences qui pourrait traiter efficacement les bases de données génomiques. Les décisions architecturales entourant Kestrel ont mené en particulier à une densité de calcul qui permet à ce système de 9 millions de transistors VLSI, d'un FPGA et de diverses puces mémoire de dépasser les performances d'un actuel poste de travail [39].

B. Schmidt et al. ont réalisé un système qui vise le traitement à haute performance de bases de données sur des nouvelles architectures massivement

parallèle [54]. Ceci vise l'atteinte de la puissance de calcul des superordinateurs à bas prix. La première architecture est réalisée autour d'une grappe d'ordinateurs liés par un réseau de communication à grande vitesse. Des tableaux parallèles à fin grain de 1024 processeurs Systola sont reliés à chaque nœud. La deuxième architecture est le Fuzion 150. C'est un ordinateur parallèle avec un tableau linéaire SIMD de 1536 éléments de traitement sur une seule puce. Les auteurs ont présenté la conception d'une application de traitement de base de données basée sur l'algorithme de S-W afin de dériver un assignement efficace sur ces architectures. Les implémentations mènent à un gain de temps d'exécution significatif pour le traitement à grande échelle de base de données. L'architecture Fuzion 150 était de 25 à 30 fois plus rapide que la grappe de processeurs Systola [54].

2.3.4 Autres accélérations

Plusieurs implémentations FPGA on été développées depuis 2003 pour d'autres algorithmes. À Berkeley, Chang a implémenté étapes 1 et 2 du système BEE2 (*Berkeley Emulation Engine 2*) [55]. Chang a exploré une nouvelle approche pour le problème d'alignement de bioséquence en utilisant un FPGA basé sur le système de calcul BEE2, qui fournit un temps d'exécution de 100~1000 fois plus rapide pour l'exécution de l'algorithme de BLAST (*Basic Local Alignment Search Tool*). Le rapport de prix-performance est de plus de 1000 fois plus bas que les solutions existantes de *cluster* de PC. Les forces de cette plateforme sont l'interaction entre la partie logicielle et la partie matérielle, et le pipelinage du réseau de flot de données.

Une autre implémentation de l'algorithme de BLAST sur d'autres plateformes non-FPGA, comme celle d'Altschul et al.. Les auteurs emploient une approche heuristique moins précise que l'algorithme de S-W, mais elle est plus rapide [57]. Des implémentations sur des grappes d'ordinateurs ont été réalisées comme mpiBLAST

de Darling et al.. Les auteurs ont effectué leur implémentation à travers la bibliothèque *MPI* [37].

D'autres projets ne suivent pas exactement les étapes de BLAST, mais utilisent des approches heuristiques qui lui ressemblent, comme le RDisk de l'université de Rennes (France) [58]. À l'université de Flinders (Australie), Gardner-Stephene et Knowles ont développé un nouvel algorithme (DASH) caractérisé par une meilleure sensibilité que BLAST, ainsi que son implémentation FPGA [59].

2.4 Choix de la technologie du support matériel

La performance des systèmes de comparaison de séquences est généralement mesurée par des millions de mises à jour de cellules d'élément de traitement par seconde (MCUPS) [60]. Le Tableau 2.1 donne un résumé de quelques systèmes mentionnés précédemment, concernant leur performance, leurs approches de conception, et leur classification technologique [38][54].

Tableau 2.1. Performance des architectures implémentant l'algorithme de S-W.

Plateforme	Année	Technologie	PEs	MCUPS
Splash II	1993	1000 nm FPGA	1536	3000?
Kestrel	1997	500 nm ASIC	512	400
Fusion 150	2000	250 nm ASIC	1536	2500
DeCypher	2001	180 nm FPGA	N/A	7500
GeneMatcher2	2001	130 nm ASIC	2872	16000

Excepté la première architecture, la comparaison des caractéristiques des systèmes présentés dans le Tableau 2.1 montre une augmentation progressive des

performances en MCUPS. La performance globale du système est très proche entre les deux technologies ASIC et FPGA, mais le grand inconvénient des ASICs est leurs coûts de conception très élevés. Par contre, les systèmes FPGA ne sont pas chers et ils sont reconfigurables.

Le potentiel des FPGAs en tant qu'accélérateurs de calcul reconfigurables résulte du parallélisme à fin grain, requit pour l'augmentation de performance de calcul [31]. Cette grande efficacité devrait produire un rendement computationnel plus élevé. La performance des FPGAs augmente d'un facteur de 4x tous les deux ans. En outre, pour les opérations utilisant des améliorations architecturales, la performance augmente d'un facteur de 5x tous les deux ans [61]. Ainsi, les FPGAs ont le potentiel d'offrir une performance globale crête (*peak*) qui pourrait atteindre les 20x tous les deux ans. Le contenu des bases de données génétiques humaines double tous les six mois [29], une augmentation de 16x tous les deux ans. Dans ces conditions, il s'avère que les améliorations technologiques des FPGA pourraient relever le défi de répondre à l'explosion des bases de données génétiques [4]. Il reste à consacrer l'effort de conception nécessaire pour y répondre d'une manière efficace.

Conclusion

Ce deuxième chapitre a présenté une revue de littérature qui a permet d'établir un état de l'art du domaine de comparaison de séquences génétiques, ainsi qu'une classification des systèmes existants. Ceci a déterminé les méthodes de traitement adoptées en logiciel, ainsi que le choix de la technologie FPGA comme support matériel de travail, vu leurs avantages discutés ci-dessus.

CHAPITRE 3 PARALLÉLISATION DE L'APPLICATION : COMPARAISON DE SÉQUENCES GÉNÉTIQUES

Dans ce chapitre, nous allons décrire notre méthodologie de parallélisation de l'application traitant la comparaison de séquences d'ADN. Nous détaillons par la suite les approches de parallélisation logicielle en utilisant les bibliothèques standard *OpenMP* et *MPI*, ainsi que les résultats obtenus. Nous montrons que cette application nécessite une parallélisation matérielle pour avoir une bonne accélération du traitement. Pour atteindre ce but, une étude de conception ainsi que trois alternatives d'implémentations sur une architecture de la technologie FPGA sont présentées.

3.1 Méthodologie

Notre méthodologie de travail consiste à déterminer les étapes permettant l'atteinte des objectifs mentionnés au chapitre Introduction. D'abord, la revue de littérature présentée dans le chapitre 2 nous a permis d'établir un état de l'art du domaine de comparaison des séquences génétiques. La taille croissante de ces données nécessite une puissance de calcul à haute performance.

Par la suite, nous avons proposé, dans un premier article [4], une classification catégorique des systèmes de traitement existants, en montrant notre intérêt aux accélérateurs matériels réalisés avec des circuits ASIC et FPGA. Nous avons déterminé leurs actuelles performances vis-à-vis le défi continuel de l'explosion de taille des bases de données génétiques. Nous avons montré aussi que la technologie FPGA pourrait satisfaire ce défi [4]. C'est pour cette raison que nous l'avons choisie comme support matériel. En outre, cette technologie est caractérisée

par sa reprogrammabilité et son coût raisonnable pour réaliser une accélération matérielle.

Nous effectuons par la suite un profilage du code de l'application pour déterminer les régions critiques du traitement. Les étapes suivantes seraient : une parallélisation du code en logicielle, mesure de résultats, puis une parallélisation matérielle. Les performances obtenues des systèmes réalisés en matériel seront présentées et analysées. Nous détaillons, ci-dessous, ces étapes de travail.

3.1.1 Approches de parallélisation logicielle

Après le profilage du code séquentiel de l'application, nous allons procéder à la parallélisation des parties critiques du code. Nous procédons à des approches qui se basent sur des techniques de division de données tout en respectant leurs contraintes de dépendance.

Nous implémentons d'abord deux approches qui utilisent la bibliothèque *OpenMP*, ces techniques sont : la division en sous blocs de traitement et la transformation unimodulaire (changement de repère des boucles de traitement). Les résultats de ces implémentations seront analysés pour déterminer leurs impacts sur les performances au niveau de l'exécution de l'application.

En suite, nous effectuons une implémentation en utilisant le standard *MPI*. L'analyse et la comparaison de ses résultats avec ceux des premières implémentations (*OpenMP*) définiront s'il est utile de penser à une implémentation hétérogène qui utilise les deux standards *OpenMP – MPI*.

3.1.2 Étude de conception et réalisations matérielles

Comme nous allons le montrer dans la suite de ce mémoire, le profilage du code séquentiel montre que la partie de comparaison constitue plus de 72% du temps d'exécution global. En plus, cette tâche se répète autant de fois qu'il y a des

séquences dans la base de données. Un important gain de performance pourrait être réalisé au niveau du rendement global du système de comparaison, en effectuant la parallélisation de cette partie essentielle de l'application. Ceci nous a incités à adopter la méthode de traitement par tableau systolique. Il sera réalisé sur du matériel (carte FPGA).

Le cœur du tableau systolique est l'unité de traitement élémentaire (PE). Nous avons effectué une étude de simplification à l'aide de considérations logiques qui concernent les variables voisines de la case qui contient le résultat d'un résultat donné. Cette étude nous a permis d'établir deux conditions déterminant la valeur à passer aux variables voisines. À partir des résultats de simulation du comparateur réalisé à la base de ces deux dernières conditions, nous avons obtenu deux nouvelles conditions, à l'aide de simplifications logiques.

Le nouveau comparateur de base est le cœur de trois unités de traitement avec différents niveaux de granularité: simple, double, et quadruple. Ces trois éléments de traitement vont permettre de réaliser trois tableaux systoliques composés, successivement, de *PE_1×1, PE_2×2*, et *PE_4×4*. Ils peuvent comparer un, deux, et quatre caractères de la séquence objectif avec le même nombre correspondant dans la séquence cible.

Un tableau systolique composé du *PE_4×4* nécessite, approximativement, une vitesse de transfert de donnée quatre fois moins grande que celle du *PE_1×1*, mais une taille de donnée quatre fois plus grande (4*2 bits au lieu de 1*2 bits). Dans le cas *du PE_2×2*, nous avons deux fois la taille des données (2*2 bits) à transférer par rapport au *PE_1×1*, mais deux fois plus rapidement que dans le cas du *PE_4×4*. Ces trois systèmes font l'objet d'une étude comparative qui analyse les performances obtenues en fonctions des coûts matériels requis.

La détermination du meilleur compromis entre taille de données et vitesse de transfert pourrait être un objectif d'étude. Ceci pourrait avoir lieu lors de la finalisation de la réalisation du système complet de calcul HPC.

3.1.3 Mesure de performances

Cette étape permet de situer les performances de nos systèmes réalisés devant ceux des systèmes similaires existants. Une première réalisation est un tableau systolique à PE simple. Ce PE permet de comparer un seul nucléotide de chacune des deux séquences comparées à chaque cycle d'horloge. Ce premier tableau systolique est réalisé sur deux familles FPGA Virtex I et Virtex II. Les résultats de performances du tableau systolique simple (*PE_1×1*) font l'objet d'un article que nous avons publié à la conférence NEWCAS 2007 [62].

Nous présentons par la suite les résultas de la réalisation des deux autres systèmes de comparaison (double et quadruple). Les performances respectives sont mesurées lors de l'implémentation. Nous discutons les apports au niveau des fréquences combinées de traitement.

Une étude critique suivra, elle concerne la validation du système qui présente un bon compromis entre coût matériel / vitesse de traitement. L'étude suivante concerne la possibilité d'avoir un système de traitement reconfigurable. Ce système pourra octroyer la fréquence de traitement combinée la plus adaptée selon la longueur des séquences comparées lors de la consultation de la base de données génétiques.

Dans ce qui suit, nous détaillons les deux parties de parallélisation logicielle et matérielle. Les approches adoptées en logiciel, ainsi que le principe de conception et les réalisations effectuées en matériel seront présentés. Nous exposons par la suite les résultats de performances obtenus et leurs analyses dans le chapitre 4.

3.2 Parallélisation logicielle

Dans cette première étude, nous allons procéder à la parallélisation du code séquentiel de l'algorithme de Smith-Waterman. Cet algorithme est implémenté en utilisant les bibliothèques standard *OpenMP* (*Open Multi-Processing*) et *MPI*

(*Message Passing Interface*). Les tests et les mesures sont réalisés sur les architectures du laboratoire DRAP (Design et Réalisation d'Applications Parallèles) de l'École Polytechnique de Montréal [64].

Nous allons proposer deux approches de parallélisation avec la librairie *OpenMP*, ainsi qu'une parallélisation utilisant la bibliothèque *MPI*. Ces approches sont basées sur des modèles de dépendances de données caractérisant l'algorithme de Smith-Waterman. Nous allons analyser les mesures du temps de réponse et l'accélération de la vitesse de traitement dans une architecture parallèle.

Nous rappelons que l'algorithme de Smith-Waterman détermine les sous séquences les plus semblables entre deux séquences (alignement local) par programmation dynamique. Ceci revient à calculer une distance représentant le coût minimal de transformation d'un segment à l'autre. Trois opérations élémentaires sont employées : substitution, insertion et suppression. Le plus petit nombre d'opérations exigées pour changer un segment à un autre est la mesure de la distance entre les segments.

Comme illustré dans la Figure 3.1 ci-dessous, chaque case (élément) de la matrice représente un résultat $S_{i,j}$ calculé à partir des trois cases voisines selon une progression diagonale du haut vers le bas. Nous rappelons plus bas le système d'équations (6) du paragraphe (1.5) qui donne les formules de calcul des résultats de l'algorithme de Smith-Waterman.

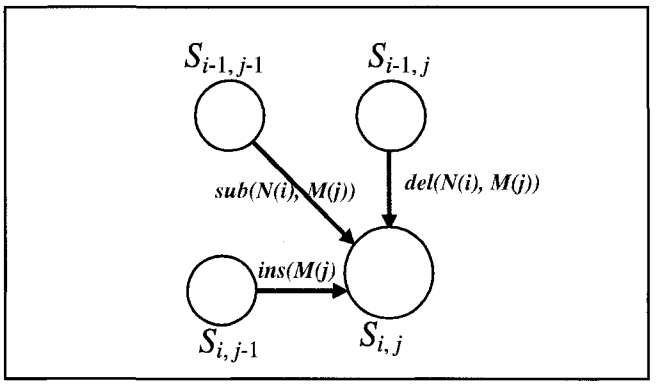


Figure 3.1 Dépendance de données entres les éléments de la matrice des résultats.

$$S_{i,j} = \max \begin{cases} S_{i-1,j-1} + sub(N(i), M(j)) \\ S_{i,j-1} + del(N(i)) \\ S_{i-1,j} + ins(M(j)) \\ 0 \end{cases} \qquad (6)$$

Nous avons restructuré le code source original [65] sous forme de fonctions indépendantes afin d'identifier clairement à quel niveau l'application consomme le plus de ressource CPU. La Figure 3.2 montre le profilage du code tout en mentionnant le temps consommé par appel de chaque fonction. À l'issu de l'application d'une batterie de tests, le Tableau 3.1 présente les données relatives à la consommation du temps CPU en prenant en compte le nombre d'appel de chaque fonction. Les mesures ont été prises sur la machine Charybde [64] au nœud 1, avec les paramètres de compilation suivants:

pgcc smithwaterman_seq.c –Mprof=func

En analysant ces premiers relevés, et après avoir calculé les ratios de temps d'exécution par rapport au temps d'exécution global, nous remarquons que le bloc de calcul principal (*calcul_score()*) consomme presque tout le temps CPU (72%). Ce bloc de calcul doit être parallélisé en tenant compte des contraintes de dépendance de

données, imposées par la programmation dynamique (Figure 3.1). Ceci nécessite la division de ces données qui respecte leur dépendance selon les approches de parallélisation que nous allons présenter dans les sections suivantes.

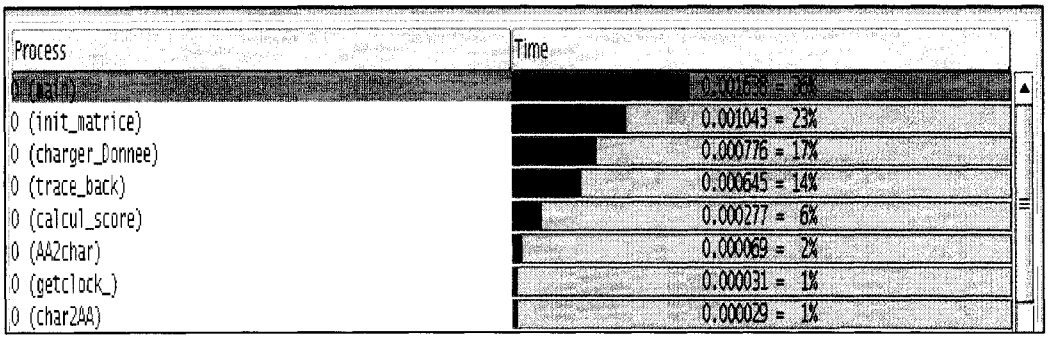


Figure 3.2 Résultat du profilage du code source séquentiel.

Tableau 3.1 Relevé du profilage de l'application S-W_seq.c en séquentiel.

Temps (%)	Coût (%) par appel	Nombre d'Appels	fonction
0.17%	36 %	1	*Main ()*
0.11%	23%	1	*init_matrice()*
0.08%	17%	1	*Charger_donnee()*
0.07%	14%	1	*Trace_back()*
72.07%	6%	2500	*Calcul_score()*
17.89%	2%	1862	*AA2char()*
9.61%	1%	2000	*Char2AA()*

3.3 Parallélisation avec le standard *OpenMP* (mémoire partagée)

L'interface logicielle *OpenMP* est un standard définissant un ensemble de directives et de fonctions. Il permet de gérer le partage du calcul entre plusieurs processeurs sur des architectures à mémoire partagée [66].

Nos approches de parallélisation, tiennent en compte les contraintes de dépendance de données induites par les relations du calcul du résultat de programmation dynamique. À titre de rappel, la programmation dynamique est une

stratégie qui se résume à la division du problème initial en petits problèmes élémentaires. Le type de dépendances de données élémentaires mentionné dans la Figure 3.1 se propage tout au long des éléments successifs du problème global.

Fa. Zhang et al. ont proposé une méthode basée sur le principe «diviser pour conquérir» [56]. Cette méthode se base sur la subdivision de la matrice des séquences d'ADN en sous blocs. Cependant les auteurs, ne mentionnent ni comment la division est effectuée, ni comment le nombre de bloc optimal à l'issu de leur division est obtenu. Dans le but de chercher une division optimale, nous avons procédé par une parallélisation qui exploite les directives dont dispose la bibliothèque *OpenMP* suivant des approches de parallélisation.

Comme vision générale, nos approches de parallélisation se basent sur la division de la matrice des résultats en sous matrices ayant une mêmes taille. Ces sous matrices vont constituer des blocs de calcul. Chaque rangée de blocs devra être calculée par un processus (*thread*) donné. Ce processus réveillera le processus suivant pour commencer le calcul du bloc en dessous. La même procédure sera suivie pour le reste des blocs jusqu'à la fin de chaque rangée de blocs de la matrice originale [63].

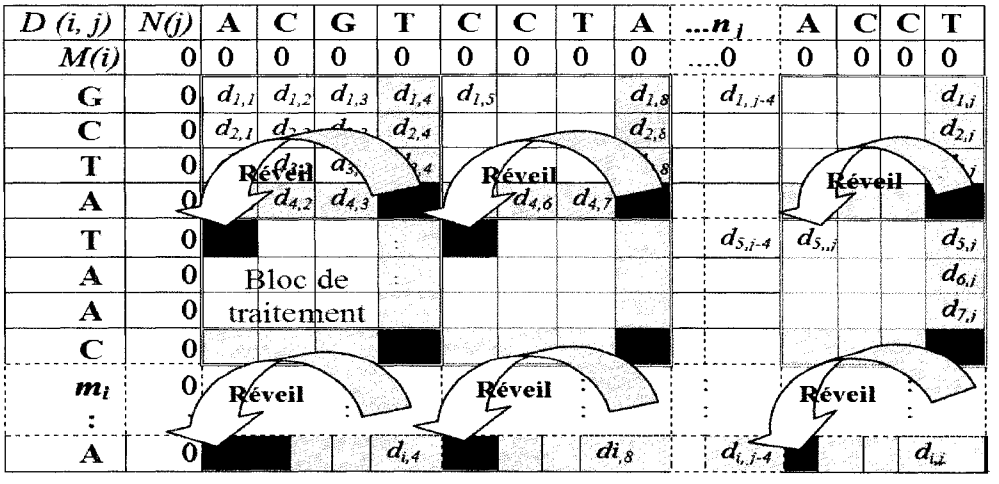


Figure 3.3 Division de la matrice des résultats en blocs de calcul avec réveil des processus.

La mise en œuvre de cette approche nécessite des modifications au niveau des boucles de parcours de la matrice des résultats. Pour ceci, deux boucles imbriquées sont utilisées. Elles parcourent l'ensemble des blocs de la matrice, sous forme de plusieurs lignes dont chacune est une rangée de blocs de calcul (Figure 3.3). La distribution des itérations du code sur les processus se fait avec la manière suivante :

- Les itérations de la boucle globale sont réparties sur plusieurs processus
- Chaque itération de la boucle interne calcule le résultat du bloc élémentaire tout en assurant une synchronisation entre les rangées successives de blocs.

Nous avons effectué deux approches de parallélisation que nous allons présenter ci-dessous.

3.3.1 Première approche

Cette approche de parallélisation consiste à diviser le tableau (matrice de résultats) en sous tableaux, avec un nombre de bloc paramétrable. L'affectation des processus est comme suit :

- Un processus commence le 1er bloc.
- lorsque le processus 1 termine le 1er bloc, il fournit les données nécessaires pour le calcul du bloc suivant. Il enclenche ainsi le processus 2 qui calcule le bloc 2.
- Avant qu'un prochain processus effectue le calcul de son bloc, il doit s'assurer que le processus calculant le bloc précédent ait terminé son traitement, avant de commencer son propre calcul. Pour ceci, un mécanisme de synchronisation entre processus a été implémenté comme le montre la Figure 3.4.

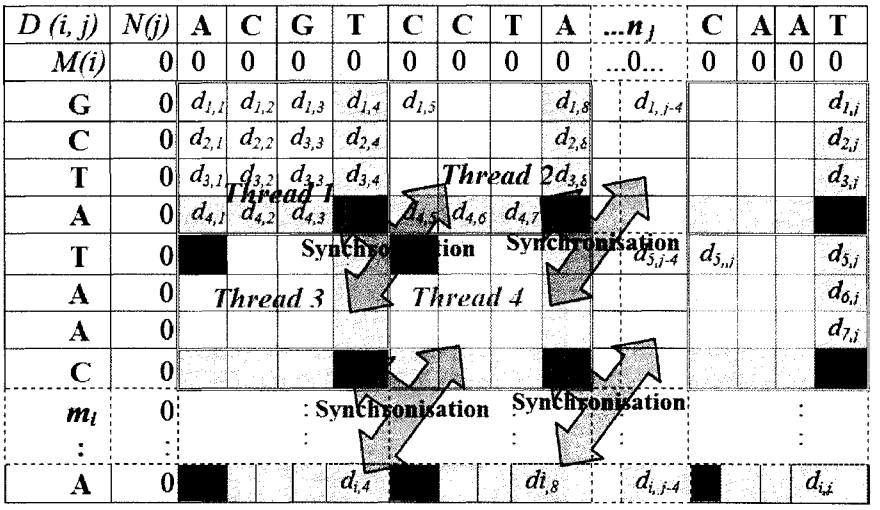
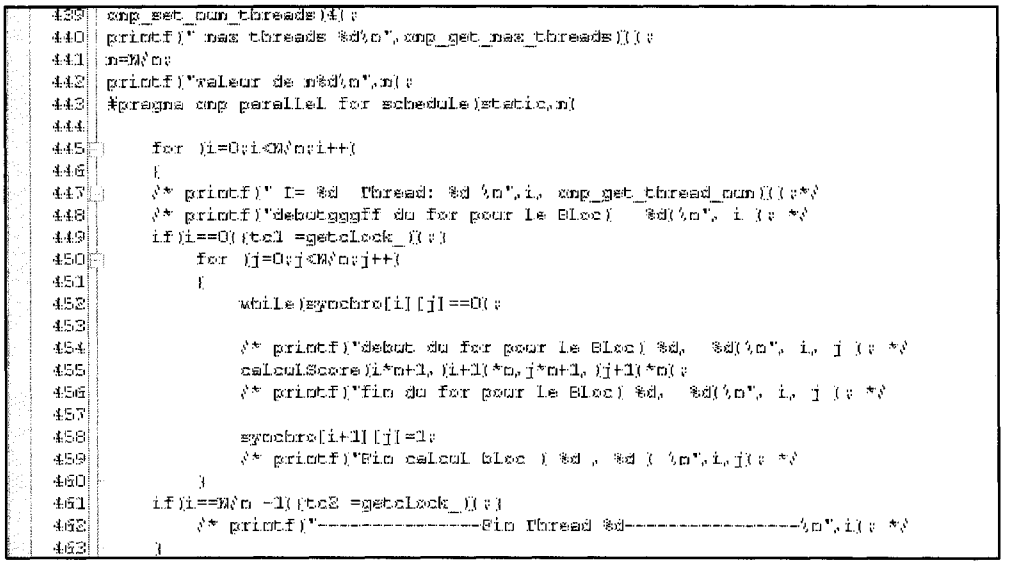


Figure 3.4 Réveil des processus et passage de données par synchronisation.

Nous avons divisé la matrice en blocs ayant une taille paramétrable, ceci pour déterminer la taille des blocs optimale pour l'architecture d'exécution cible. Une modification au code est apportée en ajoutant une matrice de synchronisation afin de contrôler le lancement des processus. Ces derniers doivent respecter la contrainte de dépendance de données. Ainsi, nous avons implémenté un mécanisme qui assure la synchronisation entre les blocs de traitement par la matrice « synchro[i][j] » (Figure 3.5). Un tel mécanisme fait défaut à la librairie *OpenMP* (elle n'en dispose pas).

Comme notre but est de chercher un impact positif sur le temps d'exécution parallèle, nous avons utilisé un pragma statique (Figure 3.5). Ce pragma effectue un octroi des processus aux blocs de calcul d'une manière statique dès le début de traitement. L'octroi dynamique des processus était aussi implémenté en mettant l'option du pragma à dynamique, comme l'illustre la Figure 3.6 ci-dessous. Ce changement d'option n'a pas apporté d'amélioration significative par rapport au temps d'exécution séquentiel. Des résultats plus détaillés sont présentés au paragraphe suivant.



```
439  omp_set_num_threads(4);
440  printf(" max threads %d\n",omp_get_max_threads());
441  m=N/n;
442  printf("valeur de m%d\n",m);
443  #pragma omp parallel for schedule(static,m)
444
445      for (i=0;i<N/n;i++)
446      {
447      /* printf(" I= %d  Thread: %d \n",i, omp_get_thread_num());*/
448      /* printf("debutgggff du for pour le Bloc)   %d(\n", i ); */
449      if(i==0){tc1 =getclock_();}
450          for (j=0;j<N/n;j++)
451          {
452              while(synchro[i][j]==0);
453
454              /* printf("debut du for pour le Bloc) %d,  %d(\n", i, j ); */
455              calculScore(i*n+1, (i+1)*n,j*n+1, (j+1)*n);
456              /* printf("fin du for pour le Bloc) %d,  %d(\n", i, j ); */
457
458              synchro[i+1][j]=1;
459              /* printf("Fin calcul bloc ( %d , %d ) \n",i,j); */
460          }
461      if(i==N/n -1){tc2 =getclock_();}
462          /* printf("--------------Fin Thread %d----------------\n",i); */
463      }
```

Figure 3.5 Parallélisation : octroi statique des processus avec synchronisation.

```
439  omp_set_num_threads(4);
440  printf(" max threads %d\n",omp_get_max_threads());
441  m=N/n;
442  printf("valeur de m%d\n",m);
443  #pragma omp parallel for schedule(Dynamic,m)
444
445      for (i=0;i<N/n;i++)
446      {
```

Figure 3.6 Parallélisation avec octroi dynamique de processus.

3.3.2 Résultats et analyse de la première approche

La Figure 3.7 montre l'accélération du code parallèle sur la plateforme Charybde [64][64], par rapport au code séquentiel de l'application. Nous avons effectué une variation de la taille des blocs (20, 250, 500, et 1000 résultats), ainsi que le nombre de processeurs effectuant le traitement.

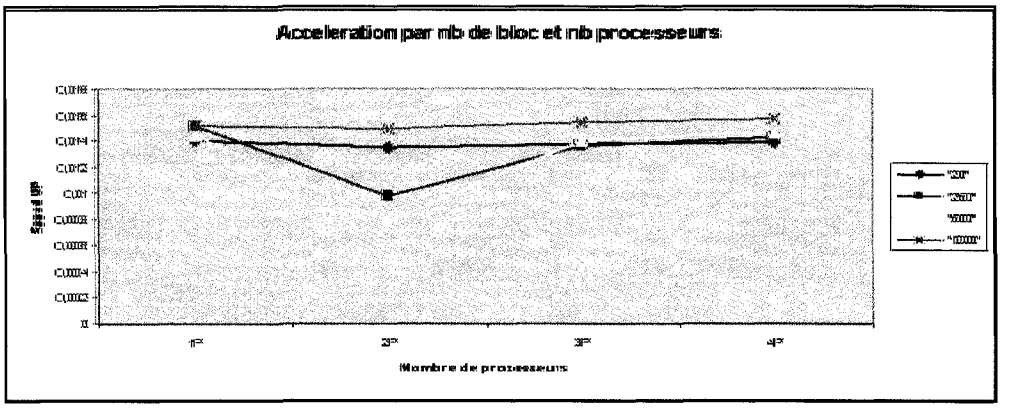


Figure 3.7 Impact de la taille des blocs sur l'accélération avec un pragma statique.

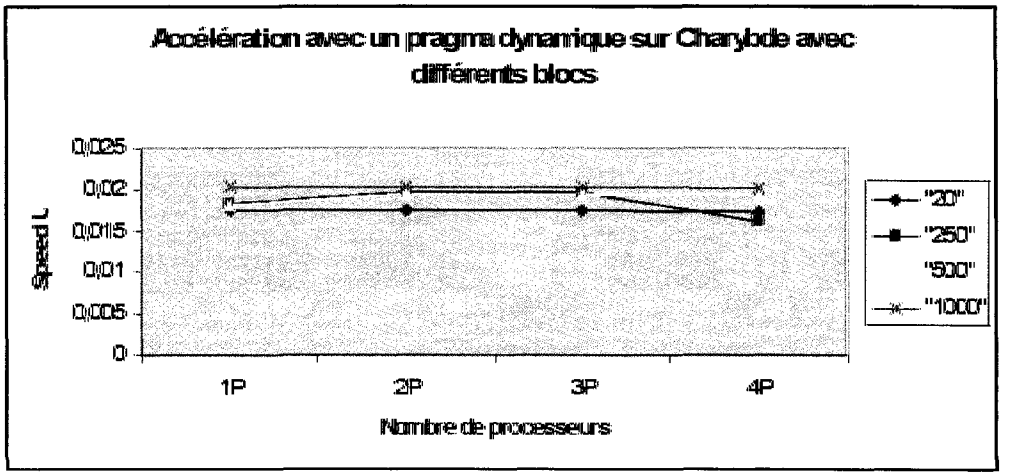


Figure 3.8 Impact de la taille des blocs sur l'accélération pour un pragma dynamique.

Dans les deux cas, pragma statique et dynamique, les mesures de temps des exécutions parallèles présentent des régressions par rapport au temps séquentiel (Figure 3.7 et Figure 3.8). Cette dégradation est due au fait que les synchronisations entre les blocs des rangées de la matrice initiale causent des blocages momentanés lors de l'exécution. L'interprétation de ce blocage est que les processus assignés aux

différentes itérations de la boucle parallèle resteront bloqués jusqu'au moment où un processus exécutera la première rangée. Si nous n'avions pas mis le paramètre *m* (nombre de processus) égal au nombre des itérations de la boucle parallèle (nombre de rangées de calcul), le résultat aurait été pire. Ce paramètre *m* est modifiable au niveau de l'option du pragma parallèle : *schedule(dynamic(ou)static,m).*

Dans le cas où le nombre de processus est inférieur au nombre de rangées de blocs (nombre d'itérations de la boucle parallèle), il pourrait y avoir un blocage au moment de l'exécution. Ce blocage est causé par le premier octroi aléatoire des processus, car il y a une espérance qu'aucun processus ne sera assigné à la première rangée. Ceci a une influence directe sur les autres processus du fait qu'ils boucleront à l'infini. L'explication de ce blocage est que les autres processus attendent leur enclenchement à la fin du traitement du premier bloc, lui-même n'était pas assigné à un processus à cause du premier octroi aléatoire.

Pour avoir des résultats significatifs, nous avons procédé par l'acquisition de la première référence temporelle à l'intérieur de la boucle parallèle avec une condition (si *i=0* : première rangée). Ceci évite un temps additionnel d'attente du lancement de l'exécution de la première rangée. La deuxième référence temporelle est positionnée à la sortie du dernier bloc de calcul. La différence de ces dernières références donnera le temps de calcul parallèle sans la prise en compte du temps de blocage. Nous l'avons appelé temps parallèle modifié. Ce temps comportait une amélioration par rapport au temps d'exécution séquentiel dans le cas d'un bloc ayant une taille de 1000 éléments (matrice entière), comme le montre la Figure 3.9. Dans le cas de partage en blocs de 20 éléments, ce temps a subi une régression (Figure 3.10).

L'analyse de la Figure 3.8 démontre que plus la taille de la matrice est grande, plus l'accélération est bonne. La même remarque peut être faite dans le cas de mesure du temps parallèle modifié (Figure 3.9 et Figure 3.10). En effet, avec une taille de bloc de 1000, qui constitue la matrice entière, on obtient un meilleur résultat par rapport au

cas d'une taille de blocs de 20 éléments. Ce constat peut être attribué à l'effet cache et aux coûts supplémentaires de mise à jour du vecteur de synchronisation.

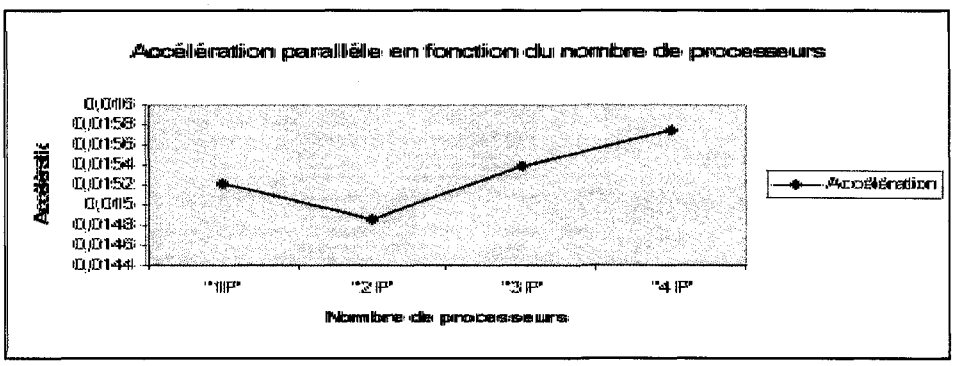


Figure 3.9 Accélération du temps parallèle modifié avec des blocs de 1000.

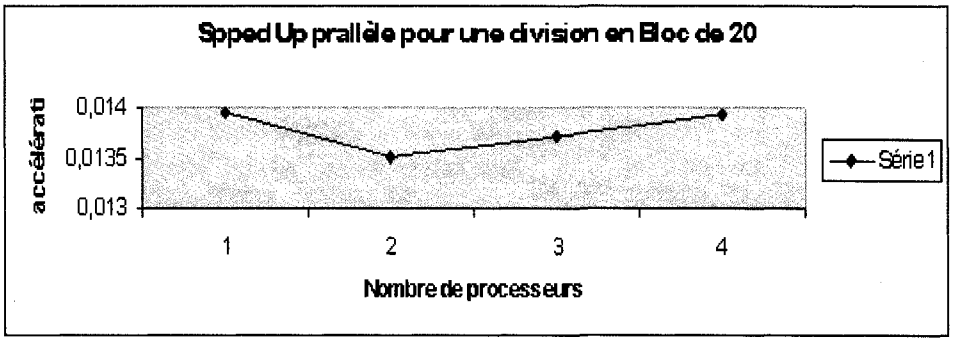


Figure 3.10 Accélération dans Charybde avec des blocs de 20 éléments.

Nous constatons donc que la bibliothèque *OpenMP* est limitée pour offrir une synchronisation *ad hoc*, car nous avons créé notre propre mécanisme pour répondre à ce besoin. Le coût supplémentaire lié au vecteur de synchronisation générant l'amorçage des processus a affectée négativement le temps de traitement parallèle en logiciel.

Les résultats obtenus ont conduit à tirer une conclusion par rapport à la parallélisation logicielle en mémoire partagée. Elle se résume à un impact défavorisant le principe de partager le calcul en blocs. Ceci est dû aux dépendances mutuelles des données entre processus nécessitant une synchronisation. Elle-même nécessite un temps additionnel de traitement et un espace supplémentaire sur la mémoire partagée.

L'amélioration du temps de calcul ne dépasse pas 1.5%. Vu ces résultats insatisfaisants de cette première approche, nous proposons une deuxième approche de parallélisation. Cette approche se base sur un changement de repère des boucles de parcours de la matrice. Ceci vise la réduction des dépendances entre les processus, au lieu de diviser la matrice principale en blocs de traitement. Le changement de repère revient à faire une transformation code-code en intégrant les directives de la bibliothèque *OpenMP*. Cette approche est présentée à la sous-section suivante.

3.3.3 Deuxième approche (transformation modulaire)

Cette nouvelle approche de parallélisation *OpenMP* se base sur une transformation modulaire, ou un changement de repère des boucles de parcours de la matrice des résultats. Nous rappelons qu'une transformation modulaire permet de transformer un code source séquentiel en un autre code. Ce dernier code a comme principales caractéristiques l'optimisation de certains paramètres, comme le parallélisme et la localité de données (minimiser les transferts de données). Cette transformation est modélisée par une matrice M entière carrée [34]. Le nouveau sens de parcours (rangées verticales du coté droit de la Figure 3.11) n'est pas sujet à une contrainte de dépendance de données. Les nouvelles boucles d'itérations obtenues permettent alors d'effectuer un traitement parallèle.

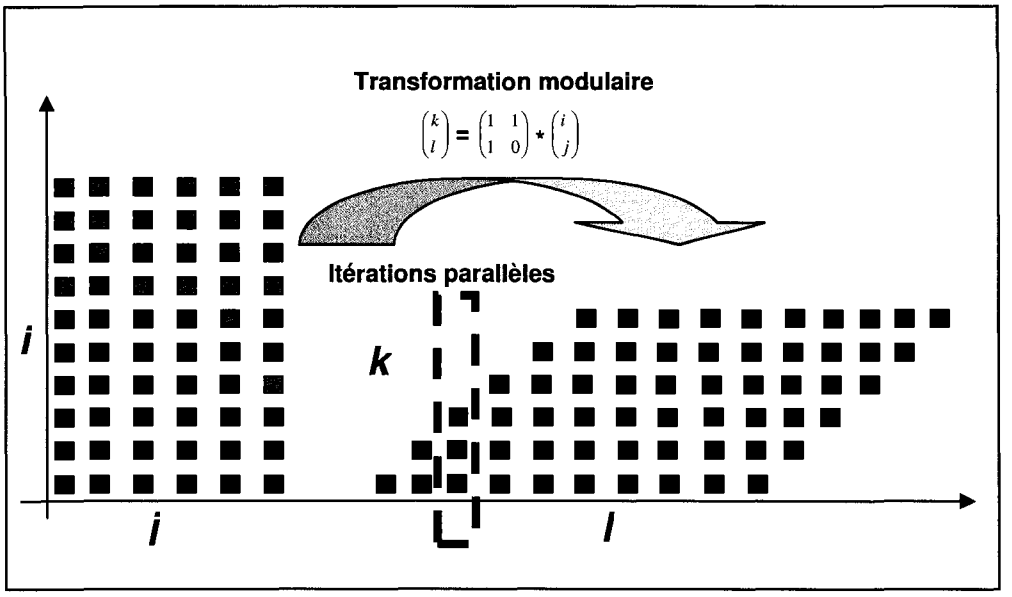


Figure 3.11 Transformation modulaire : changement de repère des boucles d'itérations.

On prend l'exemple de code suivant :

```
do i = 0 to 4
        do j=0 to 4
                S1: A[i, j]= lecture;
                S2: B [i, j]= A[i, j] + A[i-1, j-1];
        fin do
fin do
```

Comme on peut le constater dans ce code, les dépendances de *S1* vers *S2* sont $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$ et $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$. La parallélisation de ce code n'est pas possible, car le calcul de *B[i, j]* nécessite la disponibilité de la donnée *A[i, j]*.

Pour pallier au problème précédent on propose de changer l'ordre d'exécution des itérations des boucles en appliquant une transformation unimodulaire $T=\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$, qui est calculée en fonction des dépendances de données [34]. Cette transformation

est utile pour notre application, parallélisation du code de l'algorithme de Smith-Waterman, vu la similitude de la nature des dépendances de données.

Après l'application de cette transformation nous obtenons un nouveau code avec le même nombre de boucles (2 boucles), dont les bornes sont calculées de la manière suivante :

On applique la transformation T au vecteur $\begin{pmatrix} i \\ j \end{pmatrix}$ composé des itérations des boucles initiales *i* et *j*, on aura un nouveau vecteur $\begin{pmatrix} k \\ l \end{pmatrix}$ ayant comme composantes les variables d'itération des nouvelles boucles *k* et *l* :

$$\text{On a : } \begin{pmatrix} k \\ l \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} * \begin{pmatrix} i \\ j \end{pmatrix} \Leftrightarrow \begin{cases} k = i + j \\ l = i \end{cases} \quad (1)$$

D'une manière générale les bornes des boucles *i, j* sont données par :

$$\begin{cases} i\_debut \le i \le i\_fin \\ j\_début \le j \le j\_fin \end{cases} \quad (2)$$

En combinant les deux systèmes d'équations (1) et (2), on obtient le système suivant :

$$\begin{cases} (i\_debut + j\_debut) \le k \le (i\_fin + j\_fin) \\ Max(k - i\_fin, i\_début) \le l \le Min(k - i\_debut, i\_fin) \end{cases} \quad (3)$$

Où, *Max(x, y)* et *Min(x, y)* sont les fonctions qui sélectionnent le maximum et le minimum entre les variables x et y, respectivement.

La fonction qui calcule le résultat de Smith-Waterman aura deux nouvelles boucles (interne et externe). Le calcul effectué par chaque itération de la boucle

interne peut être exécuté indépendamment des calculs des autres itérations. Ceci signifie que les itérations de la boucle interne deviennent complètement parallèles, ce qui nous permet d'utiliser une directive de parallélisation *OpenMP* (boucle *for parallel*). Dorénavant, les boucles du bloc de calcul prennent une nouvelle forme présentée dans la Figure 3.12, ci-dessous.

```
On pose l = i1, k=j1,

for (i1=idebut+jdebut;i1<=ifin+jfin;i1++)
 { #pragma omp parallel for schedule(static,m)
 for (j1=MAX2(idebut,i1-jfin);j1<=Minim2(ifin,i1-jdebut);j1++)

        { i=j1;
        j=i1-j1;
        diag = h[i-1][j-1] + sim[a[i]][b[j]];
        down = h[i-1][j] + DELTA;
        right = h[i][j-1] + DELTA;
        max=MAX3(diag,down,right);
        if (max <= 0) {
        h[i][j]=0;
        xTraceback[i][j]=-1;
        yTraceback[i][j]=-1;
                        .....
        }
 }
```

Figure 3.12 Code de l'application avec des nouvelles boucle de parcours.

3.3.4 Résultats de l'approche *transformation modulaire* et analyse

Les mesures de temps d'exécution sur la machine Charybde [64], sont effectuées en variant la taille de la matrice ainsi que la taille de la région parallèle au niveau des itérations de la boucle parallèle (Figure 3.13).

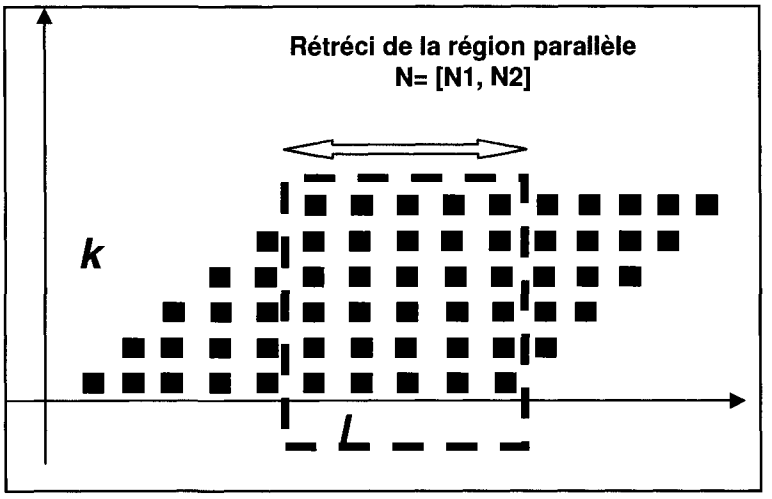


Figure 3.13 Redimensionnent de la zone parallèle.

Le Tableau 3.2 et le Tableau 3.3, ci-dessous, donnent des mesures pour des matrices de tailles (1000*1000) et (100*100) points.

Tableau 3.2 Différents temps d'exécution pour une taille de la matrice N=1000.

changement de repère	Taille de la matrice: N= 1000/ Temps de calcul (*ms*)
Séquentiel	0.123308
Tout Parallèle	1.146966
N=[200,1800] : parallèle	1.126877
N=[500,1500] : parallèle	1.015643
N=[800,1200] : parallèle	0.565284

Tableau 3.3 Différents temps d'exécution pour une taille de la matrice N=100.

changement de repère	Taille de la matrice: N= 100/ Temps de calcul (*ms*)
Séquentiel	0.115899
Tout Parallèle	0.146221
N=[20,180] : parallèle	0.156221
N=[50,150] : parallèle	0.1597750
N=[80,120] : parallèle	0.1398211

Les principales remarques sont les suivantes :

- Le temps d'exécution sans changement de repère (0.015 *ms*) est meilleur par rapport au temps d'exécution du code séquentiel avec changement de repère dans les deux cas (taille : 1000 et 100). Ceci est expliqué par le fait que le changement de repère induit un doublement des tailles des boucles d'itérations (Figure 3.13), ce qui prolonge le temps d'exécution.
- Tous les codes parallèles (pour toutes les tailles de la région parallèle) sont moins performants par rapport au code séquentiel avec changement de repère. Ceci est dû aux temps additionnels nécessaires pour la création des processus pour chaque section d'éléments à exécuter.
- Nous avons eu un temps d'exécution moins long (il s'approche du temps séquentiel) quand nous rétrécissons de plus en plus la région parallèle. C'est la région qui présente un maximum de parallélisme (Figure 3.14, Figure 3.15).

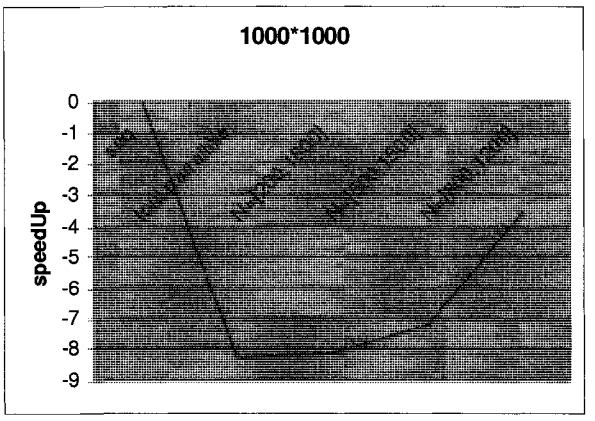


Figure 3.14 Régression de performance par rapport au code séquentiel (matrice 1000*1000).

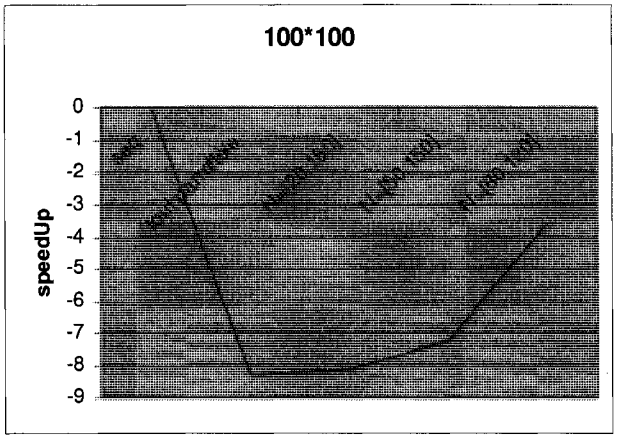


Figure 3.15 Régression de performance par rapport au code séquentiel (matrice 100*100).

Le changement de repère a produit des boucles internes parallèles, dont les performances en terme de temps d'exécution étaient réduites devant le code séquentiel. Ceci est causé par les temps de création multiple des processus pour l'exécution des différentes sections parallèles (Figure 3.14, Figure 3.15). D'après ces derniers résultats, nous pouvons conclure que le gain de performance est au niveau de la boucle externe, c.-à-d., au niveau de la grande granularité.

Ces performances auraient été meilleures si la boucle parallèle était la boucle externe, mais les dépendances de données, dans le contexte de notre application, imposent une parallélisation interne des itérations. Devant ces résultats il reste à remarquer que la parallélisation en bloc (première approche) reste relativement meilleure que celle de changement de repère en utilisant la bibliothèque *OpenMP*.

Dans la section suivante nous allons procéder à une parallélisation qui utilise la bibliothèque *MPI* uniquement, au lieu d'une parallélisation hybride *OpenMP/MPI*. Ceci est justifié par la qualité des résultats de la parallélisation *OpenMP* qui a causé une régression au temps d'exécution parallèle.

3.4 Parallélisation *MPI* (mémoire distribuée)

La bibliothèque *MPI (message passing interface)* est un standard de spécifications de passage de messages [67]. Elle permet à plusieurs processeurs (grappe de processeurs) de travailler en parallèle en mémoire distribuée.

Nous adoptons une méthode de parallélisation qui ressemble à celle de Yao Zheng et al. [56], bien que les auteurs effectuent leur traitement sur un réseau d'ordinateurs publiques (reliés par Internet). Cette méthode consiste à envoyer la ligne, la colonne et le premier élément diagonal, précédant le bloc à calculer.

3.4.1 Approche de parallélisation *MPI*

Pour cette parallélisation *MPI*, nous allons garder le concept de bloc de la première approche. Ainsi, la matrice de résultats est subdivisée en sous blocs logiques interdépendants (dépendances de données élémentaires). Le programme s'articule autour de deux types de processeurs : le processeur maître et les autres processeurs esclaves.

Le processeur maître

Le processeur maître, ou *dispatcher*, gère la matrice globale et envoie les blocs de données à calculer aux différents processeurs en fonction de leur disponibilité (Figure 3.16). Il possède en plus de la matrice globale, deux listes. La première liste permet de représenter les blocs calculables, elle est mise à jour au fur et à mesure que les blocs peuvent être calculés. La deuxième liste représente les processeurs en attente de calcul.

Voici les étapes de l'algorithme du processeur maître:

- Envoyer les données du premier bloc au 1er processeur libre.
  - Tant que le dernier bloc n'a pas été calculé :
    - Réception du bloc calculé.

- Rajout du numéro du processeur correspondant au bloc reçu à la liste des processeurs en attente.
- Mise à jour des blocs calculables.
- Ajout à la liste des blocs calculables, les blocs qui peuvent être calculés.
- Tant qu'il y a des processeurs libres et des blocs calculables :
    - Envoi du premier bloc de la liste des blocs calculables au premier processeur en attente, et ainsi de suite.

Les autres processeurs

Les processeurs esclaves reçoivent l'ordre et les données nécessaires pour le traitement désiré. Ils suivent les étapes de l'algorithme suivant:

- Recevoir les données du bloc à calculer.
- Calculer le bloc.
- Envoyer le bloc calculé au processeur maître.

La Figure 3.16, ci-dessous, illustre la procédure d'échange de données entre le processeur maître et ses processeurs esclaves.

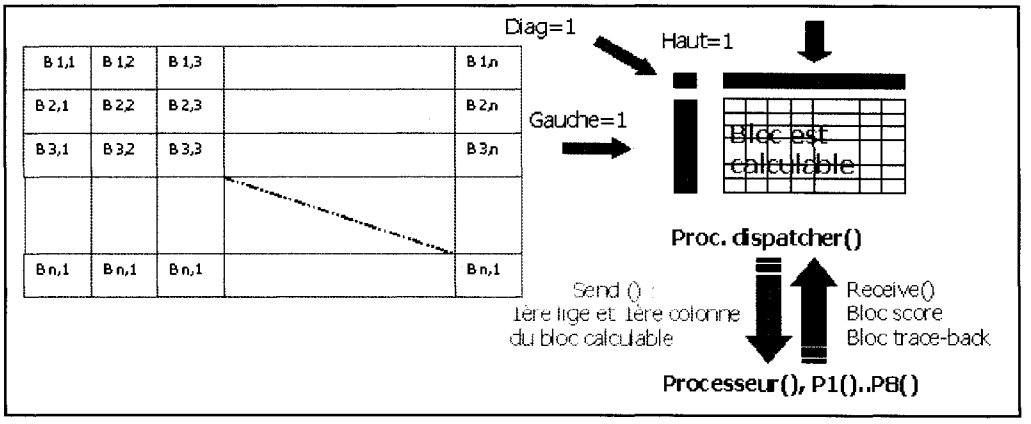


Figure 3.16 Procédure d'envoi - réception des blocs par *MPI.*

3.4.2 Résultats et analyse de la parallélisation *MPI*

Les tests ont été réalisés sur la machine Charybde MPICH-1.2.6_Myrinet_GNU [64]. Pour chaque nœud, 4 processeurs ont été réservés. Chacun des temps présentés dans le Tableau 3.4, est la moyenne des temps de deux exécutions consécutives. Le temps d'exécution de code séquentiel est : 0.076702 ms.

Nous remarquons que l'application perd de la performance lorsqu'elle est parallélisée avec *MPI*. Cela s'explique par le fait que le temps de calcul est inférieur au temps de communication. En effet, le temps de calcul par processeur atteint un maximum de 15% du temps total d'exécution lorsqu'on augmente la taille des blocs jusqu'à 1000. La Figure 3.17 illustre cet état de fait.

Tableau 3.4 Temps de calcul total en ms, matrice 1000*1000.

Taille des blocs	Nombre de nœuds							
	1	2	3	4	5	6	7	8
100	0.176	0.248	0.270	0.278	0.286	0.287	0.334	0.300
250	0.343	0.376	0.391	0.406	0.432	0.413	0.422	0.413
500	0.486	0.508	0.522	0.520	0.534	0.532	0.533	0.535
1000	0.509	0.558	0.560	0.559	0.558	0.556	0.557	0.562

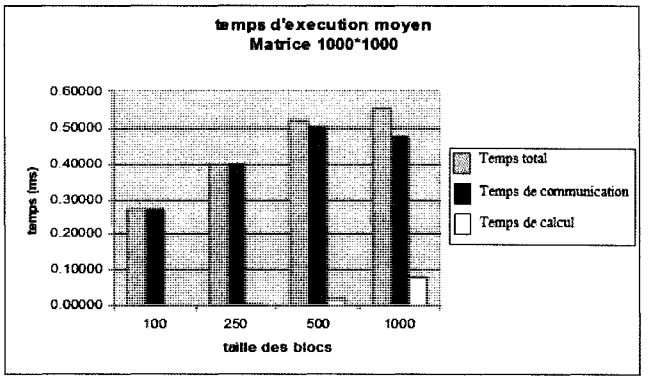


Figure 3.17 Diagramme de temps d'exécution par rapport aux communications.

D'une façon générale, on remarque que les meilleurs résultats en parallèle sont obtenus lorsqu'on exécute l'application avec le moins de nœuds possibles [68]. On observe la même allure pour les blocs ayant la plus petite taille (Figure.3.18). Ceci s'explique par le fait que sur le même nœud, le temps de communication (qui est le facteur le plus pénalisant) est minimisé quand les processeurs esclaves appartiennent tous au même nœud que celui du processeur maître. En outre, plus la taille des blocs est petite plus le nombre de régions parallèles maximales est grand.

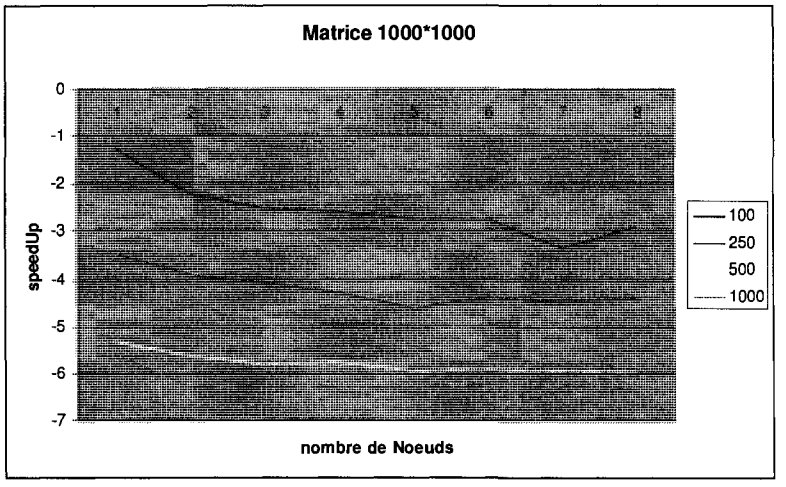


Figure.3.18 Présentation du temps d'exécution avec nombre de nœuds variable.

3.5 Apport de la parallélisation logicielle

Dans les deux dernières sections, nous avons effectué une étude de parallélisation du code de l'application de Smith-Waterman avec les bibliothèques *OpenMP* et *MPI*. Nous constatons que cette application connaît une régression de performances en exécution parallèle par rapport à l'exécution séquentielle. Ceci est principalement dû aux raisons suivantes.

Dans le cas de la parallélisation *OpenMP*, la création des processus censés effectuer le travail partagé prend beaucoup de temps, spécialement lorsqu'on a besoin de les créer au niveau de la boucle parallèle interne.

Dans le cas de la parallélisation *MPI*, le temps de communication dû au transfert de données entre les différents processeurs est largement supérieur au temps de calcul. Par conséquent, une parallélisation hybride *OpenMP/MPI* pénalisera encore plus les performances de notre application.

Un autre type de parallélisation s'impose alors, c'est la parallélisation matérielle que nous présentons dans la section suivante.

3.6 Parallélisation matérielle

Dans cette étude de conception et de réalisation, nous allons effectuer une parallélisation matérielle qui vise l'amélioration de la vitesse de traitement. Cette vitesse est étroitement liée à la complexité matérielle, car les délais de propagation des signaux en dépendent directement. Le but est de réduire la période de traitement, tout en ayant une granularité plus fine qui produit plus d'éléments de traitement.

3.6.1 Conception d'un PE Simple

Notre objectif est de raffiner la conception de l'unité élémentaire (PE) responsable du calcul du résultat de la programmation dynamique. Pour l'achever, nous avons utilisé une méthode de simplification qui se base sur la propagation de la différentiation entre les valeurs des variables traitées. Cette méthode permet de passer l'information du PE prédécesseur vers son successeur qui calcule les prochains résultats au long du tableau systolique (le tableau de comparaison). Cette succession est imposée par la contrainte de dépendance de données entre les éléments de la matrice des résultats. Nous présentons cette contrainte dans la sous-section suivante.

3.6.2 Dépendance de données

Pour calculer un des résultats *d* de la matrice de programmation dynamique, la dépendance de données impose la disponibilité des valeurs des voisins *a*, *b*, et *c* en préalable, à chaque niveau des diagonales inverses de la matrice. Ces nouveaux résultats constitueront les valeurs *b1* et *c1* du prochain niveau, ainsi que la valeur *d1* du niveau d'après. Le graphe de dépendances de données est illustré par la Figure 3.19.

Cette sorte de dépendance de données laisse entrevoir la possibilité de calculer tous les éléments situés sur les diagonales inverses de la matrice des résultats en même temps. Ceci permet donc une parallélisation du calcul sur les diagonales inverses de la matrice, à chaque cycle de traitement (Figure 3.20).

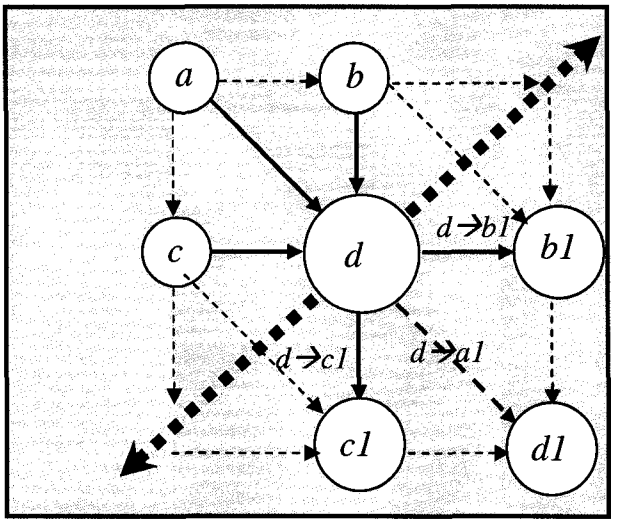


Figure 3.19 Graphe de dépendances de données pour le calcul des résultats.

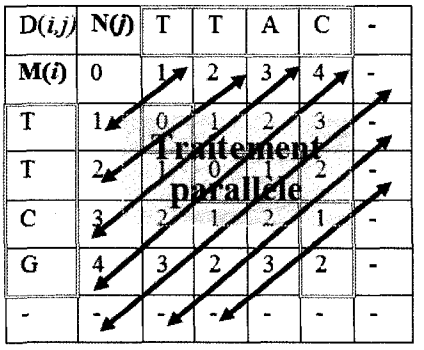


Figure 3.20 Régions du calcul parallèle dans la matrice des résultats.

Cette dépendance de données implique une connexion des éléments de traitement : prédécesseur vers successeur. Dans notre cas, on fait le lien des deux paramètres *d_out* et *b_out* à passer, successivement, au *d_in* et *b_in* du prochain élément pour le prochain cycle de traitement (Figure 3.21).

En implémentation matérielle, chaque résultat peut être calculé avec un élément de traitement. La succession des éléments de traitement en cascade permet de calculer les résultats de chaque diagonale inverse – en même temps – à chaque cycle d'horloge. Cette chaîne de PEs constitue le tableau systolique de traitement parallèle (Figure 3.21). L'interconnexion entre les éléments de traitement qui forment le tableau systolique est expliquée dans la sous-section (3.6.13).

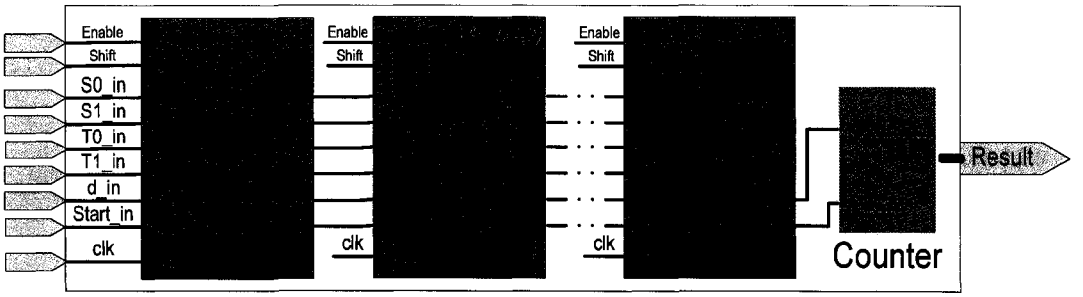


Figure 3.21 Connexion des éléments de traitement du tableau systolique.

3.6.3 Considérations logiques pour une implémentation matérielle sur FPGA

Le plus important critère pour l'amélioration de la vitesse de traitement se résume à augmenter la fréquence de travail du PE, qui est le cœur du tableau systolique. La réduction de la complexité matérielle diminuerait les délais de propagation des signaux. Ceci réduira la période de traitement, tout en permettant une granularité plus fine qui produit plus d'unités de traitement. Ce dernier paramètre représente directement le nombre de caractères, ou la longueur des deux séquences comparées, ce qui élargit le champ d'utilisation du système réalisé.

Le comparateur de base calcule la valeur d en fonction de celle de ses voisins a, b et c, données par le système d'équations (5) de la section (1.5), que nous rappelons ci-dessous.

$$d = \begin{cases} a & if\ (b = a-1)\ \ AND\ \ (c = a-1)\ \ OR\ \ (m_i = n_j) \\ a+2 & if\ (b = a+1)\ \ AND\ \ (c = a+1)\ \ AND\ \ (m_i \neq n_j) \end{cases}$$

Lipton et Lopresti avaient bien remarqué que la détermination de la valeur de d est possible en utilisant, uniquement, le bit du poids faible des variables a, b, c, et le résultat de comparaison des deux caractères m_i et n_j [22]. Ceci permet de transmettre les différentiations de ces variables au long des éléments de traitement du tableau systolique. Le signal binaire de sortie (*data_out*) du dernier élément représente une consigne à un compteur initialisé par la longueur de la séquence objectif. Cette consigne incrémente ou décrémente le compteur selon la valeur du *data_out*, ceci est illustré par la Figure 3.21.

3.6.4 Simplifications logiques

Pour simplifier notre comparateur de base, nous avons donné la valeur zéro à la variable a, le calcul de *b_out* est égal à 1 si d et c sont égaux, *d_out* est égal à 1 si d et b sont aussi égaux, sinon *b_out* et *d_out* prennent la valeur zéro. Ces deux

conditions permettent de transmettre les différentiations entre les valeurs des variables *b*, *c* et *d* des éléments de traitements successifs. Ceci est formulé par le système d'équations (7), donné ci-dessous :

$$\begin{cases} Compar = Eq \quad AND \quad (c\_in \quad AND \quad b\_in) \\ b\_out = Compar \quad XNOR \quad c\_in \\ d\_out = Compar \quad XNOR \quad b\_in \end{cases} \qquad (7)$$

où *Eq* est le résultat de comparaison de deux nucléotides n_i et m_j, déterminant la valeur de sortie (*Compar*) du multiplexeur *Mux_1*. Cette valeur détermine celles des *c_out* et *b_out* en fonction de *c_in* et *b_in*. La Figure 3.22 implémente le système d'équations (7) avec des portes logiques.

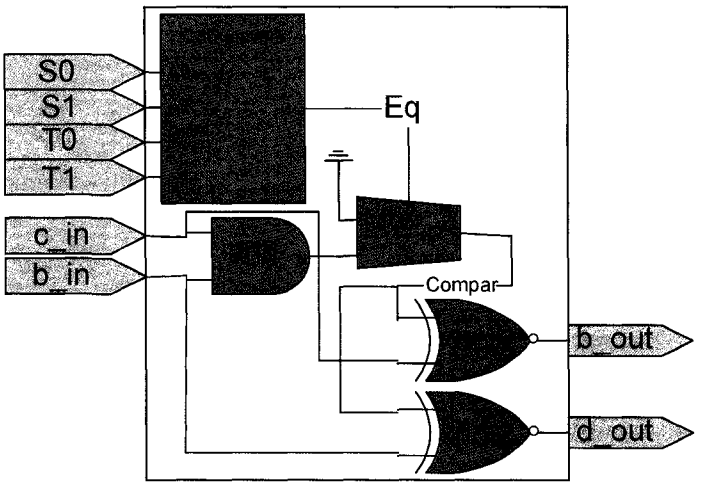


Figure 3.22 Unité de comparaison élémentaire.

3.6.5 Simplification de la conception du PE à l'aide des tables de vérité (équations logiques)

En observant la Figure 3.22, on constate que le comparateur est composé de trois niveaux logiques. Dans le but d'avoir un comparateur plus simple, nous avons

réalisé et simulé le comparateur précédent (Figure 3.22). En fonction des résultats de simulation, les nouvelles équations logiques sont établies à l'aide des tables de vérité des sorties *c_out* et *b_out* en fonction des entrées *c_in* et *b_in*. Le nouveau système d'équations résultant de la dernière procédure est le suivant :

$$\begin{cases} b\_out = \overline{Eq} \cdot c\_in + \overline{b\_in} \\ d\_out = \overline{Eq} \cdot b\_in + \overline{c\_in} \end{cases} \qquad (8)$$

Par rapport au précédent, le nouveau comparateur (Figure 3.23) est composé de deux niveaux logiques uniquement. Par conséquent, le rapport de synthèse montre une réduction des délais de propagation des signaux de 30%.

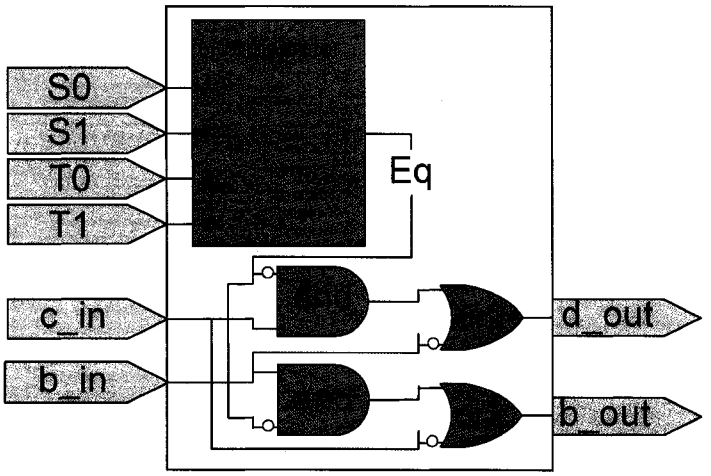


Figure 3.23 Unité de comparaison à l'aide de la table de vérité.

3.6.6 Composition d'un élément de traitement

Un élément de traitement est réalisé à l'aide d'un seul comparateur de base (Figure 3.23). Il est composé aussi d'une table de registres (*flip flops*) qui mémorisent

les valeurs des variables et les signaux de commande pour le prochain cycle de traitement. Dans cette conception, chaque élément de traitement calcule les éléments d'une colonne de la matrice des résultats (voir la section 3.6.7). En effet, le *b_out* est récupéré par l'entrée *b_in* du même PE, et sauvegardé dans un registre. Le *b_in* est utilisé au prochain cycle de traitement, car le prochain résultat se retrouve dans la même colonne de la matrice, juste en dessous du précédent. Le *d_out* est passé au *data_in* du prochain comparateur sur la même ligne de la matrice des résultats. Il est mémorisé pour le prochain cycle d'horloge. Cet élément de traitement (*PE_1×1*) permet de comparer un seul caractère de la séquence M_i avec un seul caractère de la séquence N_i, le coût matériel de ce *PE* est de 3 slices et 7 bascules (*flip flops*). La Figure 3.24 donne la structure interne du *PE_1×1*.

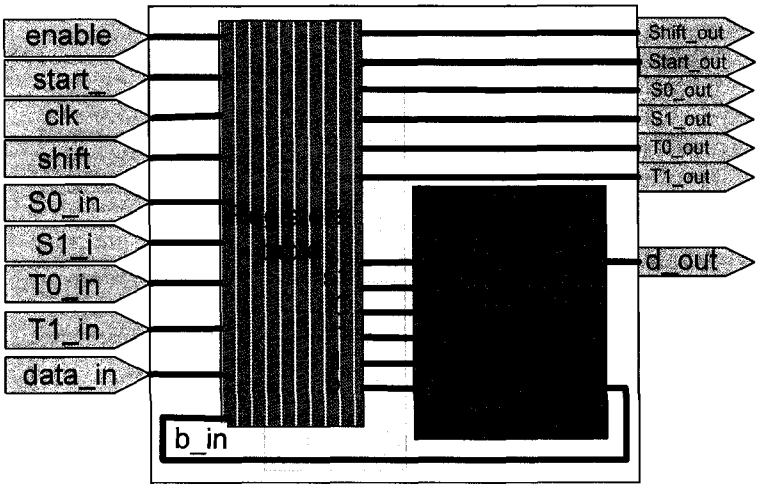


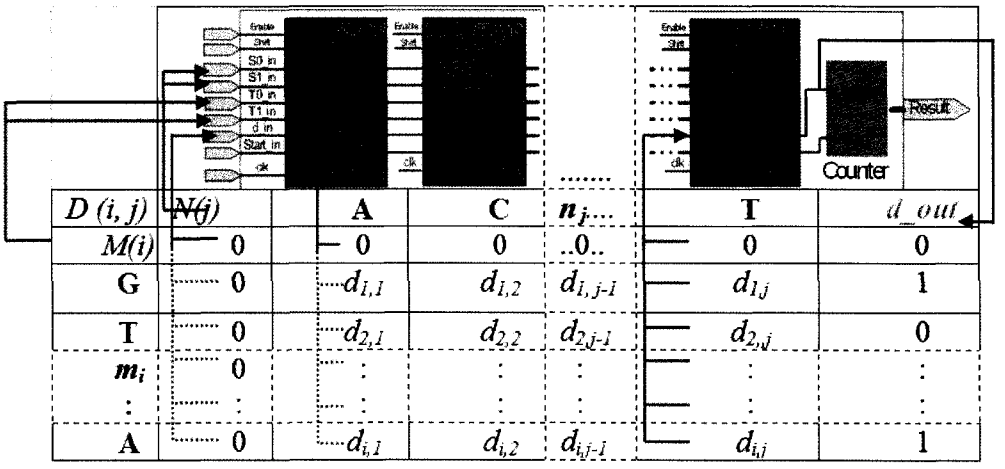
Figure 3.24 Structure interne de l'élément *PE_1x1*.

3.6.7 Traitement d'une colonne par PE, une nouvelle complexité

Comme nous l'avons montré à la section (3.6.2), la dépendance de données impose un certain ordre de traitement des éléments de la matrice des résultats. La Figure 3.20 montre que les éléments situés sur les diagonales inverses de la matrice

des résultats peuvent être calculés en même temps. Ceci peut se faire en affectant un PE pour chaque colonne de la matrice. À la première période, le résultat $d_{1,1}$ sera calculé. À la deuxième période, les deux résultats $d_{1,2}$, $d_{2,1}$ seront calculés, et ainsi de suite. Les nucléotides des deux séquences objectif ($S_{i,j}$) et cible ($T_{i,j}$) sont passés au PEs, à chaque période de traitement. Arrivé à la $n^{ème}$ périodes, les résultats calculés sont ceux situés entre la première case et la $n^{ème}$ diagonale inverse (entre $d_{1,n}$ *et* $d_{n,1}$) de la matrice des résultats (Figure 3.20). Le calcul des résultats restants, situés entre la diagonale principale inverse et la dernière case, nécessite encore n périodes de traitement. On suppose, dans ce cas, que les deux séquences ont des longueurs égales (m=n). Donc, le nombre d'opérations nécessaires pour calculer tous les résultats est 2*n (m+n dans le cas général, S et T ayant des longueurs différentes). Ce nombre d'opérations (2*n) induit une nouvelle complexité $O(n)$, au lieu de $O(n^2)$ dans le cas d'un algorithme de programmation dynamique exécuté en logiciel (voir la section 1.5).

Un autre détail concernant le calcul des résultats est que la première ligne et la première colonne de la matrice des résultats de la Figure 3.25, ci-dessous, contiennent des zéros. Ceci est dû à nos considérations de simplifications logiques. En effet, les premières valeurs de *b_in* et *c_in* commencent par prendre la valeur zéro. La valeur de *a* est prise comme référence, elle est toujours nulle (par implémentation interne du PE, voir la sous-section 3.6.4). La propagation des différentiations des valeurs des variables se propage à travers les PEs du tableau systolique (TS). Les valeurs successives du dernier *d_out* (suite de 1 et 0) constitueront une consigne pour incrémenter/décrémenter le compteur relié à la sortie du tableau systolique (dernier *d_out*).



D (i, j)	M(j)	A	C	n_j....	T	d_out
M(i)	0	0	0	..0..	0	0
G	0	$d_{1,1}$	$d_{1,2}$	$d_{1,j-1}$	$d_{1,j}$	1
T	0	$d_{2,1}$	$d_{2,2}$	$d_{2,j-1}$	$d_{2,j}$	0
m_i	0	:	:	:	:	:
:	:	:	:	:	:	:
A	0	$d_{i,1}$	$d_{i,2}$	$d_{i,j-1}$	$d_{i,j}$	1

Figure 3.25 Calcul des éléments colonnes de la matrice des résultats par les PEs du TS.

3.6.8 Interconnexion des unités de traitement : réalisation du tableau systolique

La structure uniforme du *PE_1×1* facilite l'interconnexion de ces éléments de traitement en cascade. Ceci en reliant les ports : *data_in/d_out*, *S_in/S_out* et *T_in/T_out* de chaque élément de traitement avec son successeur. Il reste à alimenter toutes les cellules par les signaux de contrôle et le signal d'horloge pour réaliser le tableau systolique, situé en haut de la matrice des résultats de la Figure 3.25.

3.6.9 Étapes de simulation

Pour effectuer la comparaison par ce dernier tableau de traitement, les quatre nucléotides sont codés en 2 bits. Les codes pour A, C, G, T sont, respectivement, 00, 01, 10, et 11. L'opération de comparaison réalisée par notre tableau systolique nécessite les procédures suivantes :

1. Effectuer un codage binaire des séquences $T_{i,i+1}$ et $S_{i,i+1}$, ex. : ACGT≡$(00011011)_2$.

2. Faire passer les caractères (codes) de $S_{i,i+1}$ dans l'ordre inverse des nucléotides dans la séquence, à partir de la référence temporelle de simulation.
3. Faire passer les valeurs de $T_{i,i+1}$ dans l'ordre direct par rapport à leurs positions dans la séquence cible.
4. Observer le résultat attendu au niveau de la sortie *data_out* durant 8 cycles, à partir du début du signal *start_out*. Ceci, car les deux séquences prises comme exemple de comparaison ont une longueur de 8 nucléotides.

Le Tableau 3.5 donne le codage des deux séquences S et T à comparer. Pour vérifier la validité des résultats obtenus par la simulation, nous donnons les valeurs des résultats calculées à partir du système d'équations (5) de la section (1.5). Ces valeurs sont présentées dans la matrice Excel du Tableau 3.6. La Figure 3.26 illustre le résultat de simulation d'ISE de Xilinx [70], où le port *data_out* contient les valeurs attendues, situés dans la dernière colonne de la matrice des résultats (Tableau 3.6). Il y a donc une conformité entre les résultats attendus (calculés) et ceux produits par la simulation de notre tableau systolique.

Tableau 3.5 Codage de deux séquences S et T et *data_out* attendu.

Séquence	Code	Séquençage à simuler								*data_out*
$S_{i,j}$ =GATATCAT	1000110011010011	3	0	1	3	0	3	0	2	01000011
T_i =GGTTATGT	1010111100111011	2	2	3	3	0	3	2	3	

Tableau 3.6 Matrice de calcul des résultats (relations (5)) et le data_out.

		G	A	T	A	T	C	A	T	data_out
	0	1	2	3	4	5	6	7	8	
G	1	0	1	2	3	4	5	6	7	0
G	2	1	2	3	4	5	6	7	8	
T	3	2	3	2	3	4	5	6	7	0
T	4	3	4	3	4	3	4	5	6	
A	5	4	3	4	3	4	5	4	5	0
T	6	5	4	3	4	3	4	5	4	
G	7	6	5	4	5	4	5	6	5	1
T	8	7	6	5	6	5	6	7	6	

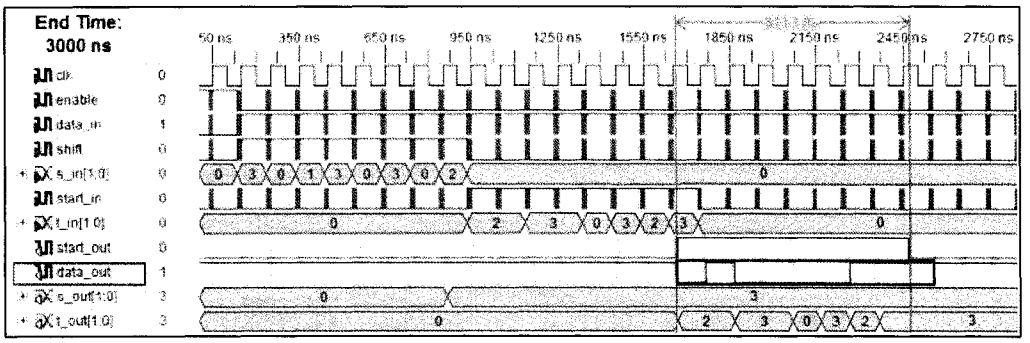


Figure 3.26 Résultats de simulation avec ISE8.2i d'un tableau systolique (16 nucléotides).

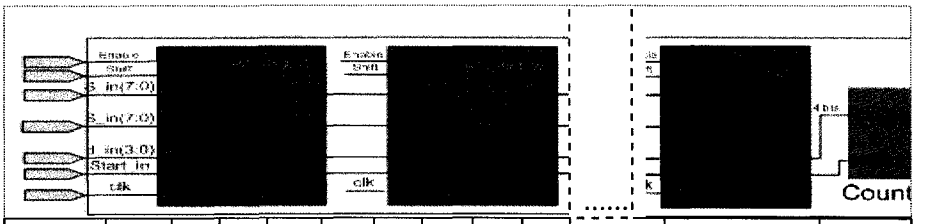
Pour cet exemple, le compteur est initialisé avec la longueur des séquences (8 nucléotides). Le *data_out* produit 5 fois la valeur '0' et 3 fois la valeur '1'. Le compteur reçoit des consignes de 3 incrémentations et 5 décrémentations. La dernière

valeur du compteur (6) est la valeur de la distance d'édition renvoyée par le tableau systolique et précalculée par la matrice des résultats (Figure 3.5). Ceci confirme l'exactitude des résultats du traitement effectué par notre système de comparaison.

3.6.10 Variation du niveau de granularité

En parallélisation logicielle, la granularité est définie comme étant le volume de traitement effectué entre deux communications (ou synchronisations). C'est une mesure qualitative du ratio entre le volume de calcul et le volume des communications [69]. Les durées de calcul sont relativement courtes par rapport aux durées des communications dans le cas d'une granularité faible. Cette granularité facilite la répartition de charge mais entraîne un surcoût important de communication. Inversement, dans le cas d'une granularité forte, il y a relativement peu de communications par rapport aux périodes de calcul. Cela donne l'opportunité d'augmenter les performances, mais la répartition de la charge est beaucoup moins évidente à mettre en place [69].

En parallélisation matérielle, ce que nous entendons par niveau de granularité, c'est la taille des blocs de résultats qu'un élément de traitement peut traiter en une seule période. À chaque période, un élément de traitement simple (*PE_1x1*) calcule un seul résultat; un élément de traitement double (*PE_2x2*) calcule un bloc de 4 résultats; un élément de traitement quadruple (*PE_4x4*) calcule un bloc de 16 résultats (Figure 3.27). La communication entre les éléments de traitement du même tableau systolique est immédiate. Mais la communication des résultats de comparaison d'un comparateur à l'autre, à l'intérieur d'un élément de traitement *PE_2x2* ou *PE_4x4*, est représentée sous forme de temps de propagation de signaux. Ceci va de l'entrée jusqu'à la sortie de chaque élément, ce qui rallonge la durée de la période de traitement dépendamment du nombre de comparateurs constituant ce PE.



$D(i,j)$	$N(j)$	A	C	G	T	C	C	T	A	$...n_j$	T				d_out
$M(i)$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	$d_{1,1}$	$d_{1,2}$	$d_{1,3}$	$d_{1,4}$	$d_{1,5}$			$d_{1,8}$	$d_{1,j-4}$				$d_{1,j}$	1
C	0	$d_{2,1}$	$d_{2,2}$	$d_{3,3}$	$d_{2,4}$				$d_{2,8}$					$d_{2,j}$	0
T	0	$d_{3,1}$	$d_{3,2}$	$d_{3,3}$	$d_{3,4}$				$d_{3,8}$					$d_{3,j}$	1
A	0	$d_{4,1}$	$d_{4,2}$	$d_{4,3}$	$d_{4,4}$	$d_{4,5}$	$d_{4,6}$	$d_{4,7}$	$d_{4,8}$					$d_{4,j}$	1
T	0	$d_{5,1}$			:	$d_{5,5}$				$d_{5,j-4}$	d_5			$d_{5,j}$	0
A	0				:									$d_{6,j}$	0
A	0													$d_{7,j}$	1
C	0				$d_{8,4}$				$d_{8,8}$					$d_{8,j}$	0
m_i	0			:				:		:			:		:
:	:			:				:		:			:		:
A	0	$d_{i,1}$			$d_{i,4}$	$d_{i,5}$			$di_{,8}$	$d_{i,j-4}$				$d_{i,j}$	1

Figure 3.27 Traitement par bloc de 4x4 éléments avec des PE_4x4.

En effet, pour le *PE_4×4,* les seize résultats $d_{i,\,4*k}$, $d_{4*k,\,j}$ (avec k compris entre *1* et *n/4*, où n est le nombre de nucléotides des deux séquences) sont calculés durant une seule période de traitement. Cette période sera relativement longue du fait que les signaux doivent passer par 4 comparateurs-lignes et 4 comparateurs-colonnes composant cet élément de traitement.

Une illustration similaire que celle du cas du *PE_4×4* peut être donnée lors de l'utilisation d'un tableau systolique composé par des *PE_2×2*. Dans ce cas, les blocs de traitement auront une taille de 4 résultats (2 éléments lignes, 2 éléments colonnes).

Il est remarquable que les périodes de travail des *PE_2×2* et *PE_4×4* seront relativement longues par rapport au *PE_1×1*, mais leurs performances seront considérées par les fréquences combinées de traitement [4]. Cette fréquence tient en compte le nombre de cellules de programmation dynamique traitées par période, elle

sera le double de la fréquence de travail dans le cas du *PE_2×2* et le quadruple dans le cas du *PE_4×4*.

Nous allons présenter la procédure de conception et de réalisation de deux tableaux systoliques constitués, successivement, par des éléments de traitement doubles, et quadruples. La discussion de leurs performances par rapport au coût matériel sera mise en évidence dans le prochain chapitre (Chapitre 4).

3.6.11 Conception et réalisation d'une unité de traitement double : *PE_2x2*

Dans le but de promouvoir la vitesse de traitement, nous avons conçu un élément de comparaison double *PE_2×2*. Il permet de comparer deux caractères de chaque séquence à la fois, ceci en utilisant quatre comparateurs de base pour effectuer le calcul de 4 résultats (2 lignes, 2 colonnes) en une seule période.

Pour simplifier la description de la structure interne du *PE_2×2*, nous allons nous positionner au premier bloc de traitement ($i=1,2$ et $j=1,2$), où les deux nucléotides comparés sont (S_1, S_2) avec (T_1, T_2). Nous nommons les comparateurs de programmation dynamique par $DP\_C_{i,j}$, ils calculent les résultats $d_{i,j}$. À l'intérieur d'un *PE_2×2*, les comparateurs $DP\_C_{1,1}$, $DP\_C_{1,2}$, $DP\_C_{2,1}$, $DP\_C_{2,2}$ comparent, successivement, les couples de nucléotides (S_1, T_1), (S_2, T_1), (S_1, T_2), (S_2, T_2). Ces mêmes couples de nucléotides sont enregistrés et seront passés au prochain *PE_2×2* à la période suivante. De la même manière, et à chaque période, les paires de nucléotides des deux séquences déjà traités, seront passées au prochain *PE_2×2;* les deux nouvelles paires de nucléotides sont chargées au niveau du premier *PE_2×2* à l'arrivée du signal *Shift*. Cette procédure permet de faire passer les deux séquences à travers les éléments du tableau systolique au fur et à mesure que l'opération de traitement s'effectue. Cette procédure doit respecter les dépendances de données des résultats précédents.

Pour les connexions externes, les deux bits du *d_out(1:0)* sont passés aux deux bits du *data_in(1:0)* du prochain élément *PE_2×2*. Les bits de contrôle sont alimentés par les signaux *enable*, *start_in*, *clk*, et *shift* à chaque période. Dans la structure interne de cet élément de traitement, le *d_out* du premier comparateur de chaque ligne est passé au *c_in* de son successeur de la même ligne. Le *b_out* est passé au *b_in* du comparateur en dessous, sur la même colonne. Les deux bits *b_out(1:0)* des deux derniers comparateurs sont mémorisés puis retournés aux deux premiers bits *b_in(1:0)* de ce même *PE_2×2*, au moment du prochain front montant de l'horloge. La Figure 3.28, ci-dessous, montre la structure interne du *PE_2×2*.

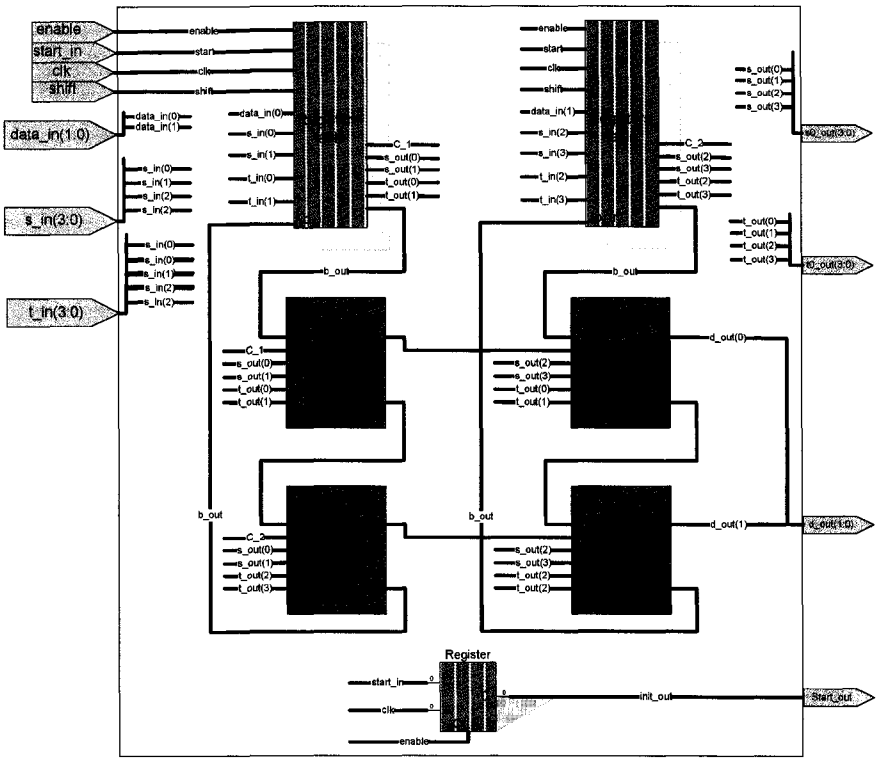


Figure 3.28 Structure interne de l'élément de traitement *PE_2×2*.

3.6.12 Conception et réalisation d'une unité de traitement quadruple *PE_4x4*

La conception de cette unité de traitement, vise le regroupement des données alimentant le tableau systolique de comparaison de séquences, ainsi que les résultats de traitement renvoyés. Quatre nucléotides de la séquence objectif sont comparés avec quatre autres nucléotides de la séquence cible située dans la base de données, à chaque période de traitement. Dans ce cas, le code des quatre nucléotides est un mot binaire de huit bits (un octet), ce qui pourrait générer une bonne cohérence au niveau entrée/sortie des données.

Cette unité de traitement élémentaire quadruple (*PE_4×4*) est similaire au *PE_2×2*, mais en plus grande granularité. Le *PE_4×4* est composé de 16 comparateurs de base qui effectuent les (4×4) comparaisons de quatre nucléotides dans chacune des deux séquences comparées.

De la même façon que nous avons présenté la description du *PE_2×2*, nous allons nous positionner aussi au premier bloc de traitement du *PE_4×4* ($i=1, 2, 3, 4$ et $j=1, 2, 3, 4$). Les quatre nucléotides comparés des séquences *S* et *T*, sont (S_1, S_2, S_3, S_4) avec (T_1, T_2, T_3, T_4). Nous nommons les comparateurs de programmation dynamique par $DP\_C_{i,j}$ qui calculent des résultats $d_{i,j}$. À l'intérieur d'un *PE_4×4*, les comparateurs $DP\_C_{i,j}$ comparent, successivement, les couples de nucléotides (S_i, T_j), avec *i* et *j* prennent des valeurs allant de 1 à 4. Les quadruples de nucléotides sont enregistrés et seront passés au prochain *PE_4×4* à la période suivante. À chaque période, de nouveaux quadruples de nucléotides des deux séquences sont chargés au niveau du premier *PE_4×4*, à l'arrivée du signal *Shift*. Ceci permet le chargement des nucléotides et de les comparer dans la même période, avec le respect des dépendances de données des résultats précédents.

Pour les connexions externes, les quatre bits du *d_out(3:0)* sont passés aux quatre bits du *data_in(3:0)* du prochain élément *PE_4×4*. Les bits de contrôle sont alimentés par les signaux *enable*, *start_in*, *clk*, et *shift* à chaque période. À l'intérieur de cet élément de traitement, le *d_out* du premier comparateur de chaque ligne est

passé au *c_in* de son successeur de la même ligne, et ainsi de suite jusqu'au *c_in* du quatrième comparateur. Au niveau de chaque colonne, le *b_out* est passé au *b_in* du comparateur en dessous. Les quatre bits *b_out(3:0)* des comparateurs de la dernière ligne sont mémorisés puis retournés aux quatre bits *b_in(3:0)* des comparateurs de la première ligne de ce même *PE_4×4*, au moment du prochain front montant de l'horloge. Cette structure interne est illustrée par la Figure 3.29.

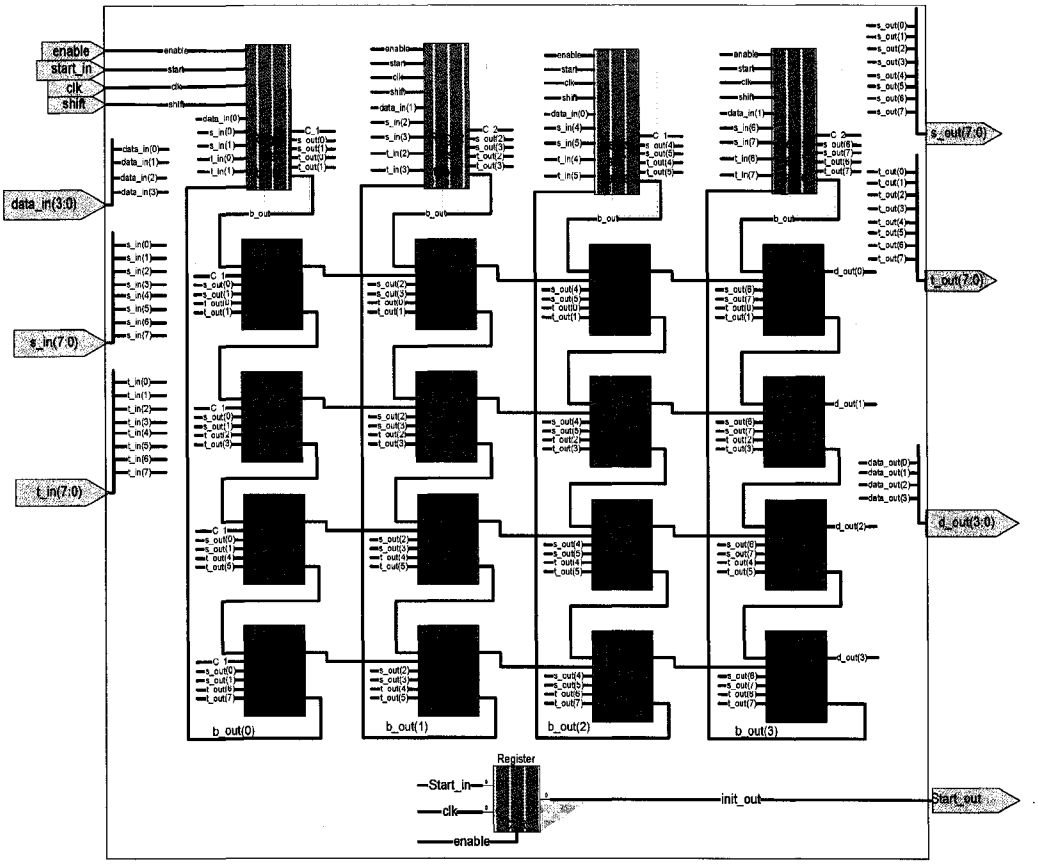


Figure 3.29 Structure interne de l'élément de traitement *PE_4×4*.

3.6.13 Principes d'interconnexion des unités de traitement : réalisation des tableaux systoliques

Dans cette conception, nous avons adopté une structure externe similaire pour les unités de traitements, même en variant leurs niveaux de granularité. La seule différence est la taille des ports d'entrées/sorties, et celle des bus de communications. Les ports *data_in/d_out(1:0)* ont une taille de 2 bits dans le cas du *PE_2×2* (Figure 3.30). Cette taille est de 4 bits, pour les ports *data_in/d_out(3:0)*, dans le cas du *PE_4×4* (Figure 3.31). Pour la taille des ports des codes des séquences, elle est de 4 bits: *S_in/S_out(3:0)* et *T_in/T_out(3:0)* dans le cas du *PE_2×2*. Dans le cas du *PE_4×4*, la taille des ports des codes des séquences est de 8 bits *S_in/S_out(7:0)* et *T_in/T_out(7:0)*. Les interconnexions internes ne sont plus visibles, il reste juste les ports : *data_in/d_out*, *S_in/S_out* et *T_in/T_out*, ainsi que les signaux de contrôle et le signal d'horloge à connecter pour réaliser le tableau systolique. La Figure 3.30 illustre la connexion de *n* éléments de traitement *PE_2×2*, cette interconnexion permet de réaliser un tableau systolique pouvant traiter deux séquences ayant des longueurs de 2**n* nucléotides.

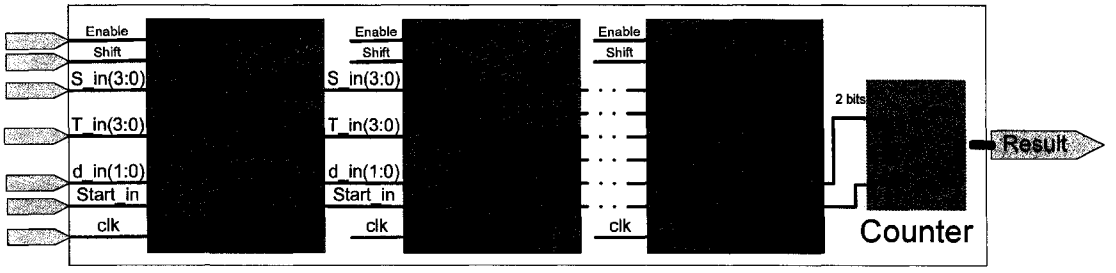


Figure 3.30 Tableau systolique réalisé par *n PE_2x2*.

La Figure 3.31 montre un tableau systolique de *n* éléments de traitement de type *PE_4×4*, pouvant effectuer la comparaison de deux séquences d'une longueur de *4*n* caractères chacune.

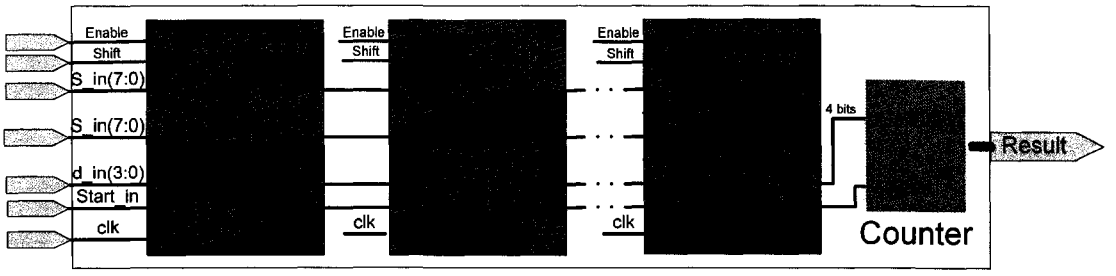


Figure 3.31 Connexion de *n PE_4×4* pour réaliser un tableau systolique.

3.6.14 Procédure et résultats de simulation du tableau systolique *PE_2x2*

La procédure de comparaison réalisée par un tableau systolique composé par des éléments de traitement *PE_2×2*, nécessite le codage des quatre nucléotides en 2 bits, ceci en regroupant les séquences en couples de nucléotides. Pour un exemple de comparaison de 16 nucléotides, les codes pour A, C, G, T sont successivement 00, 01, 10, et 11. Les procédures de comparaison sont les suivantes :

1. Effectuer un codage inverse de $T_{i,i+1}$ et $S_{i,i+1}$, c.-à-d. le code du caractère le plus à droite, prend les deux bits du poids fort (16 et 15) du mot de code [$bit_{16}bit_{15}$... bit_1], ex. : ACGT≡$(11100100)_2$=$(228)_{10}$.
2. Faire passer les caractères (codes) de $S_{i,i+1}$ dans l'ordre inverse des nucléotides dans la séquence, avec un nombre de fois égal au nombre des éléments de traitement (ex. 8 fois = 8 *PE_2×2*: pour 16 nucléotides).
3. Faire passer les valeurs de $T_{i,\,i+1}$ dans l'ordre direct par rapport à leurs positions dans la séquence cible.
4. Observer le résultat attendu au niveau de la sortie *d2_out (1 :0)* durant 8 cycles, à partir de la référence : début du signal *init_out* (Figure 3.33).

Le Tableau 3.8 donne le codage de deux séquences $S_{i,\,j}$ et $T_{i,\,j}$, prises comme exemple de comparaison. La Figure 3.33 illustre le résultat de simulation d'ISE de

Xilinx, où le port *d_out(2 :0)* contient les valeurs attendues, situées dans la dernière colonne du Tableau 3.8. Ces résultats figurent sur la dernière colonne de la matrice des résultats, dont les calculs sont effectués par le système d'équations (5) du paragraphe (1.5).

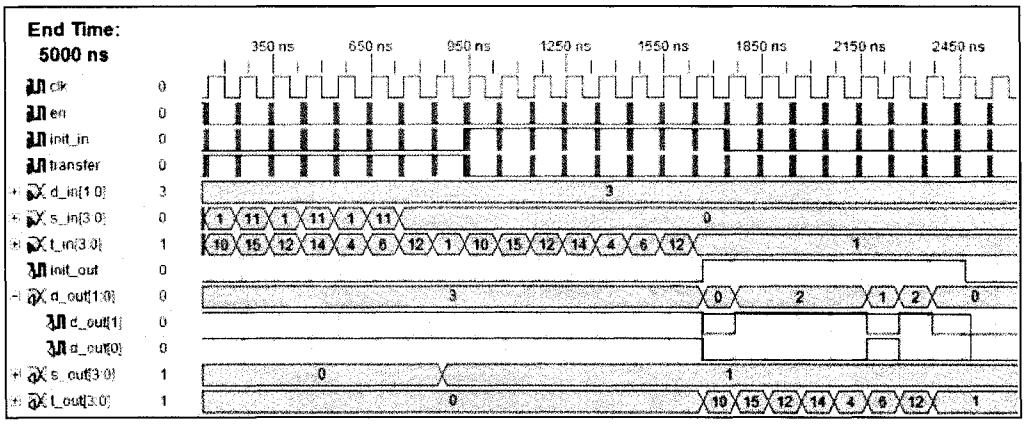


Figure 3.32 Simulation de comparaison (16*16 nucléotides) par TS *PE_2×2*.

Tableau 3.7 Codage de deux séquences *S* et *T*.

Code		D_out(3:0)
$T_{i,i+1,i+2,i+3}$	$S_{i,i+1,i+2,i+3}$	
GG=1010=10	AA=0000=0	00=0
TT=1111=15	AA=0000=0	10=2
AT=1100=12	TG=1011=11	10=2
GT=1110=14	CA=0001=1	10=2
AC=0100=4	TG=1011=11	10=2
GC=0110=6	CA=0001=1	01=1
AT=1100=12	TG=1011=11	10=2
AC=1000=1	CA=0001=1	00=0

3.6.15 Procédure et résultats de simulation du *PE_4x4*

Pour effectuer la comparaison par ce dernier tableau de traitement, les quatre nucléotides sont codés en 2 bits comme dans le cas des autres tableaux systoliques (simple, double). Les codes pour A, C, G, T sont successivement 00, 01, 10, et 11. La comparaison des deux séquences ayant une longueur de 16 nucléotides, réalisée par le tableau systolique composé par quatre *PE_4×4*, nécessite un regroupement de séquences en groupes de 4 nucléotides. Les autres procédures de simulation sont comme suit:

1. Effectuer un codage inverse de $T_{i,i+1}$ et $S_{i,i+1}$, c.-à-d. le code du caractère le plus à droite, prend les deux bits (16 et 15) du poids fort du mot de code [$bit_{16}bit_{15}$... bit_1], ex. : ACGT≡$(11100100)_2$=$(228)_{10}$.
2. Faire passer les caractères (codes) de $S_{i,i+1}$ dans l'ordre inverse des nucléotides dans la séquence, avec un nombre de fois égal au nombre des éléments de traitement (ex. 4 fois = 4 *PE_4×4*: pour 16 nucléotides).
3. Faire passer les valeurs de $T_{i,\,i+1}$ dans l'ordre direct par rapport à leurs positions dans la séquence cible.
4. Observer le résultat attendu au niveau de la sortie *d4_out (3 :0)* durant 4 cycles, à partir de la référence : début du signal *init_out* (Figure 3.33).

Le Tableau 3.8 donne le codage de deux séquences $S_{i,\,j}$ et $T_{i,\,j}$, prises comme exemple de comparaison. La Figure 3.33 illustre le résultat de simulation d'ISE de Xilinx, où le port *d_out(3 :0)* contient les valeurs attendues, situées dans la dernière colonne du Tableau 3.8. Ces résultats sont récupérés de la dernière colonne de la matrice des résultats établis par le système d'équations (5) de la section (1.5.3).

Tableau 3.8. Codage de deux séquences *S* et *T*.

Code		D_out(3:0)
$T_{i,i+1,i+2,i+3}$	$S_{i,i+1,i+2,i+3}$	
GGTT= 250	AAAA=0	1000=8
ATGT= 236	GCAT=27	1010=10
ACGC=100	GCAT=27	0110=6
ATCA=28	GCAT=27	0010=2

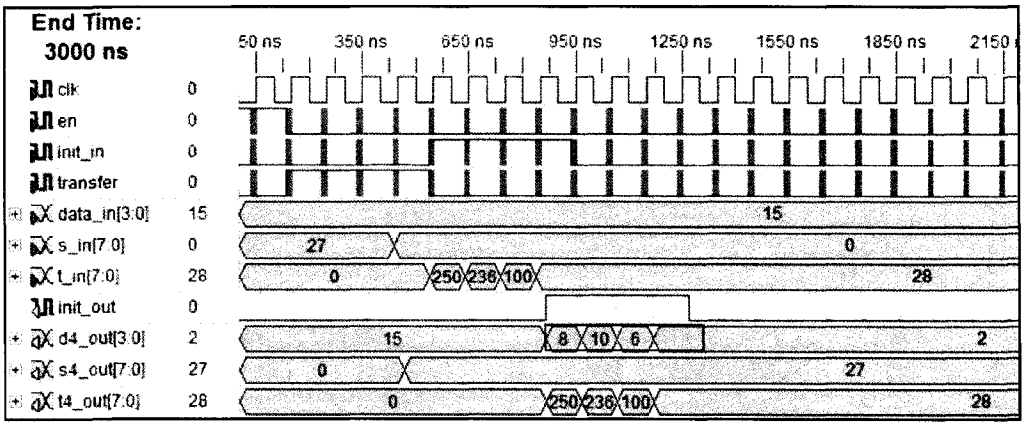


Figure 3.33 Résultats de la simulation (ISE8.2i) d'un tableau systolique (16 nucléotides).

3.6.16 Validité des résultats

Nous constatons que les trois tableaux systoliques renvoient des résultats corrects comparativement aux résultats attendus, et précalculés par le système d'équations (5). Nous avons testé notre système avec plusieurs exemples, avec des séquences cible et objectif composées par des types de nucléotides variés, tous différents, tous similaires et avec variation du nombre de nucléotides. Les résultats de tous ces tests ont montrés une conformité entre les valeurs des résultats attendus et celles renvoyés par notre système de traitement.

Ces résultats confirment la validité du traitement réalisé par les trois types de tableaux systoliques tout en variant leurs niveaux de granularité. Cette variété permettra d'étudier la possibilité d'augmenter les performances de traitement en fonction du coût matériel. Le prochain chapitre exposera les résultas obtenus ainsi que leur analyse pour ces trois systèmes.

Conclusion

Dans ce troisième chapitre, nous avons proposé deux approches de parallélisation *OpenMP* et une parallélisation *MPI*. Les résultats d'implémentation de ces approches ont montré des régressions des performances des codes parallèles. Nous avons consacré nos efforts par la suite à l'étude d'une parallélisation matérielle. Cette parallélisation a nécessité une simplification et une conception d'éléments de traitements formant des tableaux systoliques ayant des niveaux de granularité variés. Ces tableaux systoliques permettent de réduire la complexité de l'algorithme de programmation dynamique, et produisent une très grande accélération.

Nous présentons dans le dernier chapitre une mesure ainsi qu'une comparaison de performances de notre tableau systolique simple par rapport aux systèmes existants. Nous comparons aussi les performances de nos trois systèmes de comparaison à trois niveaux de granularité variables, en prenant en compte leurs coûts matériels.

CHAPITRE 4 PERFORMANCES ET ANALYSE DES RÉSULTATS

Ce quatrième chapitre présente les apports de performances obtenus par nos systèmes de comparaison implémentés à l'issu de notre étude de parallélisation matérielle. Nous présentons d'abord les gains de performances du tableau systolique à élément de traitement simple. Par la suite, nous effectuons une étude comparative entre différents tableaux systoliques, elle prend en compte la variation du niveau de granularité, le coût matériel, et l'accélération. Enfin, nous présentons notre vision de reconfigurabilité du système de traitement qui prend en compte la variation de la granularité.

4.1 Performance du *PE_1x1* par rapport aux systèmes existants

Dans ce travail de conception et de réalisation, nous avons visé l'obtention d'une amélioration de l'accélération du traitement et une réduction du coût matériel. Nos considérations logiques (paragraphe 3.6) ont conduit à la simplification de notre comparateur de base. Par conséquent, le coût matériel et la vitesse de traitement ont été positivement affectés. Notre *PE_1×1* occupe 3 slices et 7 bascules (*flip flops*) alors que, par exemple, la cellule de traitement de Yu et al. [5] occupe 4 slices et 8 bascules. Ces coûts matériels relativement réduits permettent de comparer des séquences d'une importante longueur. Elle est égale au nombre d'éléments de traitement du tableau systolique. Les taux d'occupation matérielle étaient de 98% pour les deux familles FPGA sujettes à notre implémentation.

Les performances des systèmes de comparaison de séquences sont souvent mesurées par le nombre de millions de mise à jour de cellules de programmation dynamique $S_{i,j}$ par seconde (MCUPS) [60]. Un autre paramètre de mesure de

performance est le calcul du nombre de millier de distances d'éditions globales par seconde (KDPS) [44]. Les résultats donnés dans le Tableau 4.1 représentent les performances crêtes (*peak*) des systèmes, où tous les éléments du tableau systolique sont actifs. Ceci est au niveau du calcul de la diagonale de la matrice des résultats *S*. Autrement, il y a un nombre d'éléments de traitement inactifs durant le processus de comparaison. Ce nombre d'éléments inactifs diminue en s'approchant de la diagonale dans les deux sens (du début et de la fin) de la matrice des résultats.

L'observation de la colonne des fréquences du Tableau 4.1, montre le gain de vitesse de traitement de notre système par rapport à tous les systèmes cités dedans. Ce gain varie entre 13% et 142% selon la famille FPGA considérée, ainsi que la longueur des séquences prises comme référence (2700 et 7400 nucléotides). Les deux dernières colonnes illustrent la quantité de traitement effectué. Par exemple, notre *PE_1×1×7400* pourra comparer un ensemble de 3.54 millions de séquences ayant une longueur de 7400 nucléotides, durant 1 minute uniquement.

Les résultats du Tableau 4.1 montrent que notre conception du tableau systolique présente des bons gains de performances. Ce système de traitement est composé par des comparateurs qui implémentent les relations de la formule (5) du paragraphe (1.5). Elle utilise le principe de propagation de la différence entre les valeurs de bits de données, nécessaires pour le calcul des résultats *d* de la matrice de la programmation dynamique. Nos considérations logiques ont produit des simplifications matérielles (réduction du coût matériel), ainsi qu'une accélération considérable de la fréquence de traitement du tableau systolique.

Tableau 4.1 Comparaison : performances/coûts matériels avec des implémentions existantes.

Système	Longueur de Séquence	Famille FPGA	Fréq. (MHz)	GCUPS	KDPS	Notre Accélération
JBits [23]	2700	xcv1000-6	186	502	69	+13%
Yu et al. [5]	2700	xcv1000-6	184	497	68	+14%
Notre T. Systolique	2700	xcv1000-6	210	567	78	-
JBits [23]	7400	Xc2v6000-5	293	2168	49	+48%
HokieGene [24]	7400	xc2v6000-4	180	1332	24	+142%
Notre T. Systolique	7400	xc2v6000-6	435	3219	59	-

4.2 Étude comparative de performances entre les différents éléments réalisés: coût matériel et accélération du traitement

Comme nous l'avons mentionné dans le paragraphe (3.6.5), notre *PE_1×1* occupe 3 slices et 7 bascules (*flip flops*), alors que, par exemple, la cellule de traitement de Yu et al. occupe 4 slices et 8 bascules [5]. Aussi, notre *PE_2×2* occupe 6 slices et 13 bascules, alors que le PE^4 de S. Bojanic et al. [44], traitant 2 nucléotides dans un seul cycle de traitement, occupe 7 slices et 12 bascules. Nous mentionnons que dans le cas de notre *PE_2×2,* la treizième bascule sert à propager le signale *start_in* qui enclenche le début de chargement de la séquence cible (*T*) et sa comparaison. L'opération s'effectue en même temps avec la séquence objectif (*S*), dont le chargement dépend du signal *Shift* (Figure 3.28). Ce dernier principe est valable pour toutes nos unités de traitement, et bien sûr pour les tableaux systoliques qu'ils composent. Le dernier élément de traitement, le *PE_4×4*, occupe 24 slices et 25 bascules selon les rapports de synthèses d'ISE 8.2i. Dans la littérature que nous avons consulté, il n'y a pas un système ayant un tel niveau de granularité.

Ces coûts matériels relativement réduits, permettent de comparer des séquences d'une importante longueur. Cette longueur est égale au nombre d'éléments

de traitement d'un tableau systolique composé par des éléments *PE_1×1*. Elle est égale au double dans le cas des *PE_2×2*, et égale au quadruple dans le cas des *PE_4×4*.

Sur la carte FPGA de la famille Virtex 2 : xc2v6000-6, le nombre maximal réalisé d'éléments de traitement constituant les tableaux systoliques de type *PE_1×1*, *PE_2×2*, et *PE_4×4*, est successivement, 7400, 3700, et 1050 éléments. Les taux d'occupation matérielle maximale correspondants sont de 98% pour *PE_1×1* et *PE_2×2*. Ce taux est égal à 91% pour le *PE_4×4*, comme l'indique les rapports d'implémentation d'ISE de Xilinx. La Figure 4.1 montre la grande densité d'utilisation des ressources matérielles sur cette carte FPGA. Elle montre aussi la longueur du chemin de données qui n'a pas affecté la vitesse de traitement.

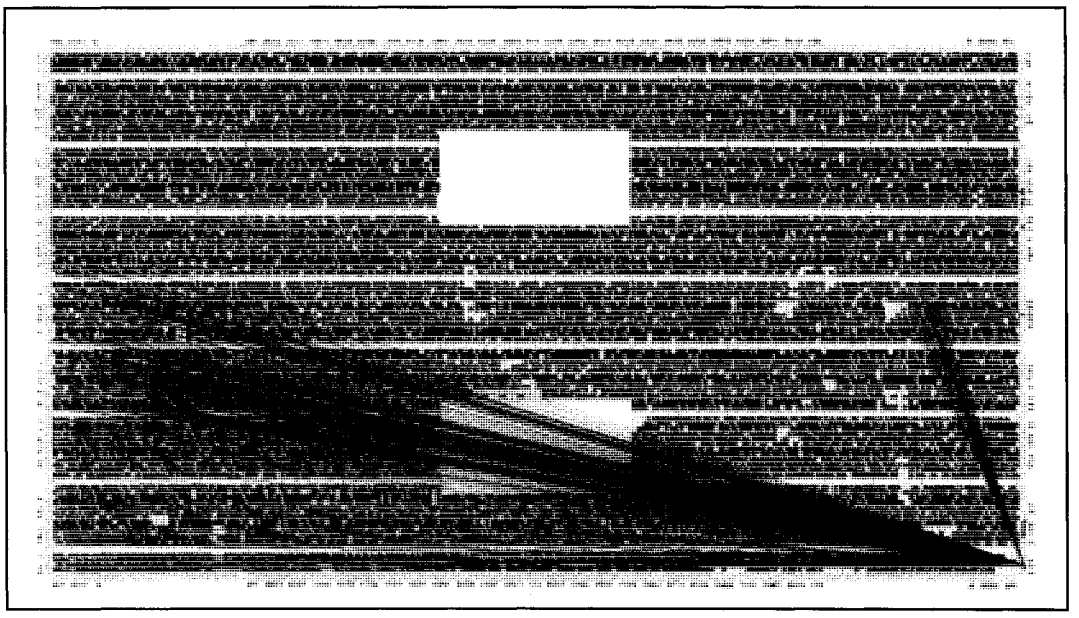


Figure 4.1 Densité d'utilisation des ressources matérielles sur la carte FPGA Virtex 2.

Nous rappelons que la fréquence combinée de traitement [4] est le double de la fréquence de travail du *PE_2×2*, car il renvoie le résultat de comparaison de deux

nucléotides des deux séquences à chaque période. Elle est le quadruple dans le cas du *PE_4×4*, car quatre nucléotides de chaque séquence sont comparés.

Le Tableau 4.2 récapitule les coûts matériels et les fréquences correspondantes (vitesses de traitements), pour les trois types de PEs réalisés.

Tableau 4.2 Récapitulatif des coûts matériels et fréquences de travail des systèmes réalisés.

Systèmes implémentés avec carte FPGA (xc2v6000-ff1517-6)		Longueur des séquences	Slices	LUTS	Flip Flops	Slices occupés	Période (ns)	Fréq. (MHz)	Fréq. combinée (Mhz)
PE_1×1	x7400	7400	33,792	33%	76%	98%	2.300	435	435
PE_2×2	x3000	6000	33,792	62%	57%	83%	4.499	222	444
	x3700	7400	33,792	76%	71%	99%	4.817	208	416
PE_4×4	x1000	4000	33,792	87%	36%	93%	8.482	118	472
	x1050	4200	33,792	91%	38%	94%	8.573	117	468

Pour le système *PE_2×2* l'amélioration de la vitesse combinée est de 2% par rapport au système *PE_1×1* pour un taux d'occupation de 83%, ce qui permet de comparer des séquences d'une longueur de 6000 nucléotides. Cette fréquence régresse de 3% pour un taux d'occupation maximal (99%) par rapport à celle du système *PE_1×1*. Mais il est possible de comparer des séquences de la même longueur maximale (7400 nucléotides) dans les deux systèmes.

Dans le cas du système *PE_4×4*, l'amélioration de la fréquence combinée est, successivement, 9% et 8% par rapport au système *PE_1×1* pour des taux d'occupation matérielle de 93% et 94%. Les longueurs des séquences comparées correspondantes sont 4000 et 4200 nucléotides. Ce tableau systolique n'est pas

comparé avec d'autres systèmes de granularité quadruple, car il n'en existe pas dans la littérature que nous avons consultée.

Au point de vue occupation de ressources, la Figure 4.2 montre que le système *PE_2×2* est le plus équilibré, car pour un taux d'occupation maximal, l'utilisation des ressources mémoire-logique est 71% - 76%. Cette occupation est moins équilibrée dans le cas du *PE_1×1* (76% - 33%), et 38% - 91% dans le cas du *PE_4×4.*

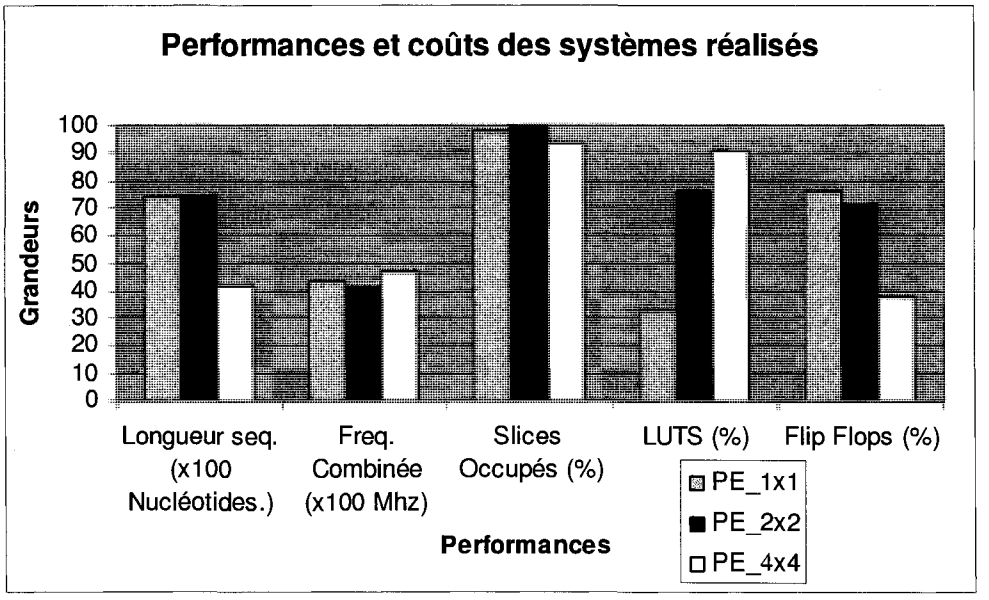


Figure 4.2 Présentation des performances et coûts des systèmes réalisés.

4.3 Détermination des points de Pareto de l'espace de conception

La performance de notre système de comparaison prend en considération trois critères : coût matériel, longueur des séquences comparées et vitesse de traitement. Pour déterminer le niveau de granularité le plus performant, nous faisons appel à la notion d'efficacité de Pareto [71]. Dans l'espace de conception, un point est appelé un *point de Pareto* s'il n'y a pas un autre point (dans cette espace de conception) avec au moins une fonction objective inférieure. Tous les autres points sont inférieurs ou

égaux. Un point de Pareto correspond à l'optimum global dans l'espace monodimensionnel d'évaluation de la conception [72].

La Figure 4.3 donne une représentation des trois niveaux de granularités, simple, double, et quadruple, ayant le maximum de performance dans chaque cas. Pour un taux d'occupation matérielle maximal, nous donnons la vitesse combinée de traitement en fonction de la longueur des séquences comparées.

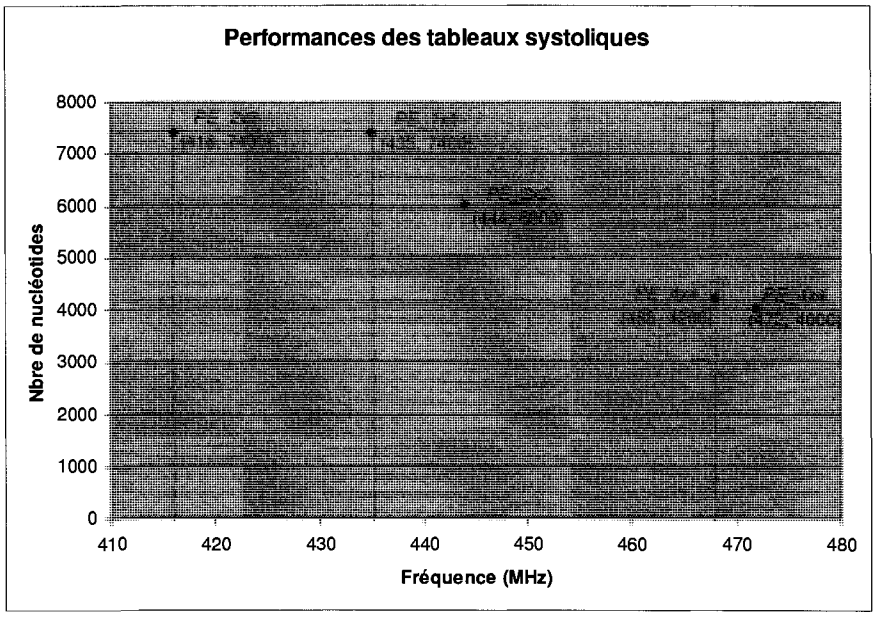


Figure 4.3 Performance par rapport au niveau de granularité.

Dans la Figure 4.3, ci-dessus, nous observons que pour les trois niveaux de granularité, les deux points (435, 7400) et (468, 4200) sont des points de Pareto, par contre le point (416, 7400) ne l'est pas. Avec ces considérations, le *PE_1×1* est retenu pour traiter des longues séquences, le *PE_4×4* est retenu pour traiter des séquences moins longues avec une plus grande vitesse de calcul.

4.4 Validation du choix de l'unité de traitement la plus performante

L'étude comparative du paragraphe précédent montre que notre conception d'éléments de traitements présente des bons gains de performances, grâce aux considérations logiques que nous avons introduit. Ceci s'est traduit par une réduction du coût matériel ainsi que des accélérations des fréquences combinées des PEs. Nous optons pour un choix conditionnel de la composition du tableau systolique selon le contexte d'utilisation, déterminé par la longueur de la séquence objectif.

Pour les séquences ayant des longueurs autour de 7400 nucléotides, l'unité de base serait le *PE_1×1*. Pour des séquences moins longues (autour de 6000 nucléotides) l'unité de base serait le *PE_2×2*. Le *PE_4×4* serait meilleur dans le cas des séquences d'une longueur autour de 4000 nucléotides. Cette souplesse de choix de système pourrait être réalisée en mémorisant les programmes configurant la carte FPGA dans une mémoire indépendante. Un simple bouton permettrait de charger le programme implémentant le type du tableau systolique voulu. Le choix sera effectué en fonction de la longueur de la séquence objectif à comparer avec les séquences cibles, situées dans la base de données.

Conclusion

Les résultats présentés dans ce dernier chapitre montrent des bons gains de performances avancés par nos systèmes de comparaison, conçus lors de notre parallélisation matérielle. Nos simplifications logiques ont produit une réduction du taux d'utilisation des ressources matérielles, ainsi qu'une accélération de la vitesse de traitement.

Notre tableau systolique basé sur un élément de traitement simple (*PE_1×1*) a présenté une accélération qui varie entre 13% et 142% comparativement aux

réalisations existantes. Ce tableau systolique a la plus grande vitesse de traitement pour la plus grande taille des séquences (7400 nucléotides) qu'on peut implémenter sur les cartes FPGA considérées. Les deux autres tableaux systoliques à niveau de granularité double et quadruple ont des gains de performances spécifiques. Le tableau systolique double (*PE_2×2*) dépasse légèrement celui du *PE_1×1* en vitesse de traitement, mais permet de comparer des séquences moins longues (6000 nucléotides). Ce système est le plus équilibré du point de vue utilisation de ressources matérielles (mémoire-logique). Le tableaux systolique quadruple (*PE_4×4*) présente un gain de vitesse de 8% par rapport au système à éléments simples, mais permet de comparer des séquences d'une longueur de 4200 nucléotides.

La variation de granularité donnerait le choix d'un système reconfigurable qui présenterait la meilleure vitesse de traitement, pour une marge de longueurs de séquences donnée.

CONCLUSION GÉNÉRALE ET TRAVAUX FUTURS

Les applications de comparaison de séquences présentent un défi de calcul HPC, vu que les algorithmes utilisés dans ce domaine requirent une grande puissance de calcul des systèmes de traitement. Nous avons fourni une classification des approches d'analyse utilisées, ainsi qu'un état de l'art des systèmes existants. Un aperçu élargi des accélérations dédiées à ce domaine a été présenté. Nous avons mentionné aussi l'effort considérable de conception qui demeure nécessaire au développement du niveau système.

Une méthodologie de travail pour atteindre les objectifs tracés a été détaillée. Nous avons réalisé deux types de parallélisation. Une parallélisation logicielle utilisant les deux standards de programmation parallèle, sur une grappe de processeurs, *OpenMP* (*Open Multi-Processing*) et *MPI* (*Message Passing Interface*) été effectuée. Les approches de parallélisation basés sur ces standards ont produit plutôt des régressions de la vitesse de traitement. Ceci est dû à deux principales raisons. La première est le manque de mécanisme de synchronisation entre les processus appelés par les directives de la bibliothèque *OpenMP*. La deuxième est le temps pénalisants des communications des messages de la bibliothèque *MPI*.

Devant ces résultats, nous avons consacré nos efforts à la parallélisation matérielle. Dans ce cadre, une étude de conception été présentée, elle a été basée sur la simplification du résultat de programmation dynamique. Des précisions relatives à la nature des résultats renvoyés ont été mentionnées (alignement local, alignement global). Nous avons conçu par la suite l'unité élémentaire responsable du calcul du résultat.

La méthode de conception utilisée se base sur la propagation de la différentiation entre les valeurs des résultats précédents, vers l'unité suivante qui calcule le prochain résultat, au long du tableau systolique. Nous avons établi deux nouvelles conditions déterminant la valeur à passer aux voisins vertical et horizontal,

à l'aide de simplifications logiques. Le nouveau comparateur de base est le cœur de trois unités de traitement à niveau de granularité variable : simple, double, et quadruple. Ces comparateurs peuvent comparer un, deux et quatre caractères de la séquence objectif, avec le même nombre correspondant dans la séquence cible, durant une période de traitement.

Les résultats obtenus à l'issu de la parallélisation matérielle montrent des bons gains de performances avancés par nos systèmes de comparaison. Nos simplifications logiques ont produit une réduction du taux d'utilisation des ressources matérielles, ainsi qu'une accélération de la vitesse de traitement. Une étude comparative des ces caractéristiques a été réalisée.

Le tableau systolique à élément de traitement simple (*PE_1×1*) a présenté une accélération qui varie entre 13% et 142% comparativement aux réalisations existantes. Par exemple, notre *PE_1×1×7400* pourra comparer un ensemble de 3.54 millions de séquences ayant une longueur de 7400 nucléotides, durant 1 minute uniquement. Ce tableau systolique a la plus grande vitesse de traitement pour la plus grande taille des séquences (7400 nucléotides) qu'on peut implémenter sur les cartes FPGA considérées. Les gains de performances des deux autres tableaux systoliques, à niveau de granularité double et quadruple, étaient spécifiques. Le tableau systolique double (*PE_2×2*) dépasse légèrement celui du *PE_1×1* en vitesse de traitement, mais permet de comparer des séquences moins longues (6000 nucléotides). Ce système est le plus équilibré au point de vue utilisation de ressources matérielles (mémoire-logique).

Le tableaux systolique quadruple (*PE_4×4*) présente un gain de vitesse de 8% par rapport au système à éléments simples, mais permet de comparer des séquences d'une longueur de 4200 nucléotides seulement. Ce tableau systolique nécessite plus de ressources logiques que de ressources de mémorisation.

La variation de granularité donnerait le choix d'un système reconfigurable qui présenterait la meilleure vitesse de traitement pour une marge de longueurs des séquences donnée.

Travaux futurs

La réalisation du système complet pourrait faire l'objectif des travaux futurs et de ce qui en découle. Principalement, le rendement global optimal qui prend en compte l'hétérogénéité *HW/SW* du système et le choix du média de communication. Un flot de conception système pourrait contenir les étapes illustrées par la Figure C.1 suivante.

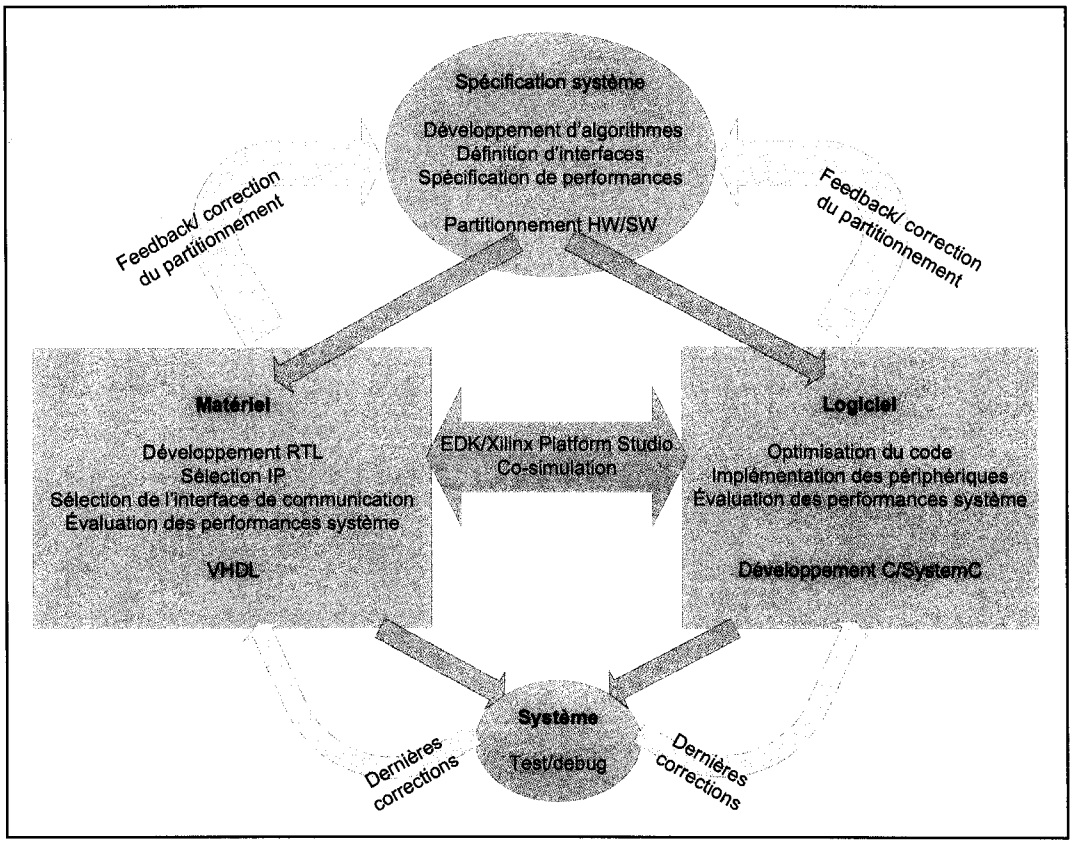


Figure C.1 Flot de conception système.

L'approche de parallélisation qui serait adoptée, pourrait être réutilisée dans d'autres domaines et types d'applications ayant des contraintes similaires. Elles concernent la dépendance de données, la quantité de données et la vitesse requise de traitement.

Une autre tâche envisageable serait la validation du partitionnement HW/SW adapté au calcul HPC, ou l'approche système. Après avoir effectué les opérations de comparaison, il faudrait effectuer un chemin arrière (*trace-back*) puis afficher les résultats de l'alignement, ce qui nécessite une implémentation de ces tâches en logiciel. Ceci pourrait se faire soit à l'aide d'une *IP* dans la carte matérielle, ou dans la station hôte qui communique avec cette carte. Un fonctionnement cohérent de l'ensemble de ces tâches requiert un bon partitionnement *HW/SW* [73].

Le choix du média de communication serait à prendre en considération, en l'occurrence sa capacité de transfert et sa latence. Ceci évite les étranglements de flux de données transmises, ce qui donne un fonctionnement systèmes cohérent entre récupération, traitement et renvoi des données résultantes.

Une autre future tâche possible, serait la validation puis la généralisation de l'approche de parallélisation. La détermination du meilleur rendement global donnerait une vision claire sur le meilleur partitionnement *HW/SW* et son interfaçage, pour ce genre d'application de calcul HPC. Ceci donnera des règles sur le meilleur niveau de granularité et le meilleur taux de transfert de données qui prend en compte leur codage. Aussi, un meilleur bus de communication qui doit prendre en compte la vitesse de traitement. Ces derniers paramètres permettent d'effectuer une validation de cette approche de parallélisation. Elle devrait produire un meilleur compromis entre puissance de calcul et vitesse de communication.

Par la suite, une généralisation et/ou une formulation de l'approche de traitement parallèle souhaitée serait possible. Elle rendra possible la réutilisation de cette approche vers d'autres applications ayant le même type de dépendance de données et nécessitant une performance de calcul similaire (HPC). Ces applications

peuvent avoir des liens avec la bioinformatique, ou de type différent comme des applications en multimédia (traitement d'images ou des séquences vidéo en temps réel).

Une dernière tâche envisageable serait l'augmentation de la signification biologique des résultats. Ceci devrait se faire en étudiant la simplification de l'autre formule de programmation dynamique, la formule de Smith-Waterman (alignement local). La simplification recherchée devrait correspondre à une implémentation optimale sur carte FPGA (codage de variable en fonction du nombre de bits, minimisation d'utilisation de ressources matérielles, etc.). Elle donnera aussi la possibilité de varier les valeurs de pénalités utilisées dans les formules des résultats. Ceci permettrait de déterminer les valeurs qui produisent des résultats avec de meilleures significations biologiques.

A l'issue de ces étapes proposées, il serait possible d'avancer un nouveau système de comparaison de séquences génétiques caractérisé d'une grande force de calcul HPC, ainsi qu'une bonne qualité des résultats renvoyés (significations biologiques). En l'occurrence, cette qualité prend en compte la précision de ces résultats, et la possibilité d'avoir une grande taille des données traitées (séquences comparées) avec leurs contraintes de dépendance de données. Un tel système pourrait à la fois répondre au défi d'explosion de taille des bases de données génétiques, ainsi qu'à l'accélération et la facilité des études statistiques biologiques. Spécialement, les études qui visent le suivi des évolutions historiques de l'information génétique, en permettant le dépistage des maladies, comme le cancer par exemple.

RÉFÉRENCES

[1] Définition de la bioinformatique, site web : http://www.pasteur.fr/recherche/unites/Binfs/definition/bioinformatics_definition.html

[2] N.C. Jones and P.A. Pevzner, *AN INTRODUCTION TO BIOINFORMATCS ALGORITHMS*. MIT Press, Cambridge, Massachusetts London, England, 2004.

[3] D.W. Mount, *BIOINFORMATICS: SEQUENCE AND GENOME ANALYSIS,* Second Edition. Cold Spring Harbor Laboratory Press, New York, 2004.

[4] N. Hireche, J. M. P. Langlois, and G. Nicolescu, "Survey of biological high performance computing: algorithms, implementations and outlook research," *Canadian Conference on Electrical and Computer Engineering*, Ottawa, Ont., Canada, 2007.

[5] C.W. Yu, et al., "A Smith-Waterman systolic cell," *Proceedings of the Tenth International Workshop on Field Programmable Logic and Applications (FPL'03),* pp. 375-384, 2003.

[6] L'ADN, site web : http://fr.wikipedia.org/wiki/Acide_d%C3%A9soxyribonucl%C3%A9ique

[7] Institut européen de chimie et de biologie, site web : http://www.cellbiol.net/ste/alpgleevec2.php

[8] Définition du génome, site web : http://fr.wikipedia.org/wiki/G%C3%A9nome

[9] Projet Comparaison du génome, France, site web : http://www.grid-france.fr/projets/comparaison-du-genome/

[10] Information du projet du génome humain, site web : http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml

[11] Institut bioinformatique européen, site web : http://www.ebi.ac.uk/

[12] Centre de l'information biologique et banque de donnée japonais, site web : http://www.ddbj.nig.ac.jp/

[13] Centre national de l'information biotechnologique, USA, site web : http://www.ncbi.nlm.nih.gov/collab/

[14] Nucleic Acids Research, Oxford journal, site web: http://www.oxfordjournals.org/our_journals/nar/about.html

[15] Programme de recherche FASTA, site web : www.ebi.ac.uk/fasta33/

[16] Programme de recherche BLAST, site web : http://www.ncbi.nlm.nih.gov/blast/Blast.cgi

[17] S. Needleman, C. Wunsch. "A general method applicable to the search of similarities in the amino acid sequences of two protein," *J. Mol. Biol.*, 48, 1970, p. 443-453.

[18] T. Smith and M. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, pp. 195-197, 1981.

[19] Richard Ernest Bellman, *AN INTRODUCTION TO THE THEORY OF DYNAMIC PROGRAMMING.* RAND Corporation, 1953.

[20] T. K. Vintsyuk, "Speech discrimination by dynamic programming," *Cybernetics and Systems Analysis*, vol. 4, pp. 52-57, 1968.

[21] M. B. Gokhale and P. S. Graham, *RECONFIGURABLE COMPUTING: ACCELERATING COMPUTING WITH FPGA*, Springer, Dordrecht, The Netherlands, 2005.

[22] R. Lipton and D. Lopresti, "A systolic array for rapid string comparison," *Hill Conference on VLSI*, 1985.

[23] S. Guccione and E. Keller, "Gene matching using Jbits," *Proceedings of Field-Programmable Logic and Application Conference*, pp. 1168-1161, 2002.

[24] K. Puttegowda et al., "A run-time reconfigurable system for gene-sequence searching", Proceedings of *16th International Conference on VLSI Design*, 2003.

[25] K. Regester, et al., "Implementing bioinformatics algorithms on Nallatech-configurable multi-FPGA systems," *Xcell Journal Online*, March 2005.

[26] M. Gokhale, et al. "Building and using a highly parallel programmable logic array," *Computer*, vol. 24, pp. 81-89, Jan. 1991.

[27] B. Schmidt, et al., "A run-time reconfigurable system for gene-sequence searching", *16th International Conference on VLSI Design*, 2003.

[28] ITRS, "PROCESS INTEGRATION, DEVICES, AND STRUCTURES," in *International Technology Roadmap for Semiconductors,* 2003 Edition.

[29] GenBank, "GenBank growth statistics," http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html, 2007.

[30] C. Johnson and J. Welser, "Future processors: flexible and modular," *CODES+ISSS '05: Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2005.

[31] K. Morris, "Saving Supercomputing with FPGAs," *FPGA and Structured ASIC Journal*, 2005.

[32] D. Bacon, S. Graham, O. Sharp. "Compiler Transformations for High-Performance Computing", *ACM Computing Surveys*, December, 1994, p. 345-420.

[33] Antoine Fraboulet. Optimisation de la mémoire et de la consommation des systèmes multimédia embarqués. *Thèse de doctorat, INSA de Lyon*, 119 pages, Novembre 2001.

[34] Michael E. Wolf, "Improving Locality and Parallelism in Nested Loops", *Ph.D. thesis, Stanford University, Computer Systems Laboratory*, August, 1992.

[35] S. Pellicer, N. Ahmed, Y. Pan, and Y. Zheng, "Gene sequence alignment on a public computing platform," *Parallel Processing, 2005. ICPP 2005 Workshops. International Conference Workshops on*, June 2005.

[36] C. Steven Mark, "Memory-hierarchy management," Rice University, 1993.

[37] Darling, A., Carey, L., et Feng, W. "The design, implementation, and evaluation of mpiBLAST". *In Proc. of CWCE,* 2003.

[38] S.F. Smith and J.F. Frenzel, "Bioinformatics searches using a single-chip shared-memory multiprocessor," *International Conference on Parallel and Distributed Processing Techniques and Applications,* 2003.

[39] A.D. Blas et al., "The UCSC Kestrel parallel processor," *IEEE Transaction on Parallel and Distributed Systems,* Vol. 16, N. 1, Jan. 2005.

[40] Paracel Inc. "Paracel's GeneMatcher for accelerating genomic data analysis installed at Schering-Plough Research Institute". *Business Wire*, USA, 1999.

[41] TimeLogic Corp., www.timelogic.com, Jan. 2007.

[42] D. P. Lopresti, "P-NAC: A Systolic Array for Comparing Nucleic Acid Sequences," *Computer*, vol. 20, pp. 98-99, 1987.

[43] D. T. Hoang, "Searching genetic data base on splash 2", *FCCM'93, IEEE Workshop on FPGA for Custom Computing Machines* (Napa, California), 1993, pp. 185-191.

[44] S. Bojanic, G. Caffarena, C. Pedreira, and O. Nieto-Taladriz, "High speed circuits for genetics applications," 2004.

[45] R. P. Jacobi, M. Ayala-Rincón, L. G. A. Carvalho, C. Llanos and R. Hartenstein, *Reconfigurable Systems for Sequence Alignment and for General Dynamic Programming*. Genetics and Molecular Research, 4(3):543-552, 2005.

[46] Y. Yamaguchi, Y. Miyajima, T. Maruyama, and A. Konagaya, "High speed homology search using run-time reconfiguration [human genome sequencing]," *12th International Conference on Field-Programmable Logic and Applications,* Springer-Verlag, London, UK, 2002.

[47] Leiserson, Charles E.; Saxe, James B. "Retiming synchronous circuitry". *HP Labs Technical Reports SRC-RR-13*. August 20, 1986.

[48] W. Nicholas, M. Yury, P. Yatish, and W. John, *Post-placement c-slow retiming for the Xilinx Virtex FPGA*. Berkeley, CA: FPGA'03, 2003.

[49] S. Dydel and P. Bala, "Large scale protein sequence alignment using FPGA reprogrammable logic devices," *Field Programmable Logic and Application,* Springer Berlin / Heidelberg, 2004.

[50] T. Van Court and M. C. Herbordt, "Families of FPGA-based algorithms for approximate string matching," *Application-Specific Systems, Architectures and*

Processors proceeding, 15th IEEE International Conference on IEEE, Computer Society, Washington, DC, USA, 2004.

[51] E. T. Chow, J. C. Peterson, M. S. Waterman, T. Hunkapiller, and B. A. Zimmermann, "A systolic array processor for biological information signal processing," in *Proceedings of the 5th international conference on Supercomputing*. Cologne, West Germany: ACM Press, 1991.

[52] P. Guerdoux-Jamet and D. Lavenier, "SAMBA: hardware accelerator for biological sequence comparison," *Comput. Appl. Biosci.*, vol. 13, pp. 609-615, 1997.

[53] T. Han and S. Parameswaran, "SWASAD: an ASIC design for high speed DNA sequence matching," Bangalore, India, 2002.

[54] B. Schmidt, H. Schroder, and M. Schimmler, "Massively parallel solutions for molecular sequence analysis," *Proc. Int'l Parallel and Distributed Processing Symp.*, pp. 186-192, Apr. 2002.

[55] C. Chang, "BLAST implementation on BEE2", Electrical *Engineering and Computer Science, University of California at Berkeley*, report 2004.

[56] Yao Zheng; Pellicer, S.; Ahmed, N.; Yi Pan, "Gene sequence alignment on a public computing platform", *Parallel Processing, International Conference Workshops on , vol., no.*, pp. 95-102, 14-17 June 2005.

[57] Altschul, S.F., Gish, W., Miller, W., Myers, E.W. & Lipman, D.J. (1990) "Basic local alignment search tool." *J. Mol.* Biol. 215:403-410.

[58] D. Lavenier, S. Guyétant, S. Derrien, and S Rubini. "A reconfigurable parallel disk system for filtering genomic banks", *ERSA'03, Engineering of Reconfigurable Systems and Algorithms*, Las Vegas, Nevada, USA, 2003.

[59] P. Gardner-Stephen and G. Knowles, "DASH: localising dynamic programming for order of magnitude faster, accurate sequence alignment," *2004 IEEE Computational Systems Bioinformatics Conference (CSB'04)*, pp. 732-735, 2004.

[60] R. Hughey, "Parallel Sequence Comparison and Alignment," *1995 IEEE International Conference on Application-Specific Array Processors (ASAP'95)*, p. 137, Jul. 1995.

[61] A. Schnore, M. Devlin, "SC05 OpenFPGA BOF", www.Openfpga.com, Nov. 2005.

[62] N. Hireche, J. M. P. Langlois, and G. Nicolescu, "A Systolic Array for Sequence Comparison based on Two-Logic-Levels Processing Element," *NWSCAS/NEWCAS*, 5-8 Août 2007, Montréal, Qc, Canada.

[63] N. Hireche, O-C. Keita, N. Kerzazi, rapport de laboratoire #1 du cours INF6601 *traitement parallèle,* "parallélisation en mémoire partagée", *École Polytechnique de Montréal*, Hiver 2006.

[64] Laboratoire du Design et réalisation d'applications parallèles, *École Polytechnique de Montréal*, site web : http://www.polymtl.ca/drap/infrastructure/index.php

[65] Code source séquentiel de l'application de Smith-Waterman, site web : http://www.cs.bc.edu/~clote/ComputationalMolecularBiology/smithWaterman.c

[66] Spécification de la bibliothèque *OpenMP*, site web: http://www.openmp.org/blog/specifications/

[67] Le standard Message Passing Interface, site web: http://www-unix.mcs.anl.gov/mpi

[68] N. Hireche, O-C. Keita, rapport de laboratoire #3 du cours INF6601 *traitement parallèle,* "Parallélisation OpenMP/MPI de l'algorithme de Smith-Waterman pour la comparaison des séquences biologiques", École Polytechnique de Montréal, Hiver 2006.

[69] Centre d'études techniques maritimes et fluviales, France, site web : http://www.cetmef.equipement.gouv.fr/projets/transversaux/cluster/calculs.html

[70] ISE Xilinx, site web: http://www.xilinx.com/ise/logic_design_prod/webpack.htm

[71] Efficacité de Pareto, site web : http://en.wikipedia.org/wiki/Pareto_efficiency

[72] Giovanni De Micheli, *Synthesis and Optimization of Digital Circuits*: McGraw-Hill Higher Education, 1994.

[73] Grant Martin, Brian Bailey, Andrew Piziali, *ESL DESIGN AND VERIFICATION: A Prescription for Electronic System Level Methodology*, Morgan Kaufman (Elsevier), San Francisco, February 2007.

ANNEXES

Annexe 1. Article publié, CCECE/CCGEI, 7-10 Mai 2006 à Ottawa

represent alignments of one sequence with spaces. $S[0, 0]$ represents the alignment of two empty sequences, and is set to zero. All other entries are computed with the following formula [7]:

$$S[i, j] = \max \begin{cases} S[i-1, j-1] + sub(N[i], M[j]) \\ S[i, j] + del(N[i]) \\ S[i-1, j] + ins(M[j]) \\ 0 \end{cases}$$

A trace-back matrix is used to keep track of the moves in the scoring matrix. Alignments are produced by starting at the highest-scoring position in the scoring matrix and following a trace path from those positions up to a box that scores zero.

For two sequences of lengths m and n, there are $(n+1) \times (m+1)$ positions to be computed. The algorithm thus has a time complexity of $O(n^2)$. It has also quadratic space complexity because the entire matrix must be kept in memory.

2. Bioinformatics algorithms implementation considerations

One of the main goals of designers of biological high performance computing is to develop a platform for efficient biological sequence analysis and especially for the SW algorithm. High-performance sequence analysis often relies on straightforward parallelization of the $O(n^2)$ DP algorithm. A common mapping is to assign one processing element (PE) to each character of the query string, and then to shift the database through the linear chain of PEs. A set of PEs can form a reprogrammable and flexible accelerator when these compromises are required. The implementation of the SW algorithm always presents a system design challenge to improve biological high performance computing using different analysis approaches.

Moore's Law observes that the number of transistors that can be integrated doubles every 18 months [8]. However, the amount of genomic data at GenBank, an annotated collection of all publicly available DNA sequences, is doubling every six months [9]. The growing gap between the amount of information to be analyzed and integration density implies that different implementation strategies must be considered. This requires a massive design effort to choose the best compromise between three criteria: flexibility, programmability and computational density.

Flexibility implies the possibility to reuse hardware for different applications. Programmability is a system-level attribute pertaining to its ability to support reconfiguration as required. Computational density is a hardware/software attribute referring to the amount of computation per area, volume, or power. It can correspond inversely to the price per unit of performance.

Considering these three criteria, four common types of processing systems can be used for sequence alignment processing [10]:

- *General-purpose supercomputers* are the most flexible means of fast sequence analysis, but they suffer from very high cost.
- *Single-purpose processors* can achieve the highest performance for a single algorithm for prices in the low tens of thousands of dollars.
- *Programmable co-processors* strive for the algorithmic flexibility of reconfigurable systems and the speed and density of single-purpose systems. Cost and ease of programming generally fall between the other two types. ASIC-based accelerators have the disadvantage of low production volume, and therefore the design costs cannot be spread over very many units. Examples of this type of accelerator are Kestrel [11], and GeneMatcher 2 [12].
- *Reconfigurable Hardware* includes systems based on FPGAs. They tend to have a much lower processing element density than single-purpose processors, but they can be faster than supercomputers. FPGA and related compiler technology is improving and FPGA-based accelerators are much less expensive to design. An example of a recent FPGA-based accelerator is the proprietary DeCypher [13], on which little detailed design information has been published.

3. Hardware architectures review and comparisons

In this section we present five hardware architectures designed specifically for sequence analysis.

The first one is Splash, described by M. Gokhale et al. [14]. A programmable linear logic array, it bridges the gap between the traditional fixed-function VLSI systolic array and the more versatile programmable array. Splash received a 1989 Gordon Bell Prize honorable mention for timings on a problem that compared a new DNA sequence against a library of sequences to find the closest match. The systolic array consists of many stages connected in a one-dimensional array. Each stage has three components: a Xilinx FPGA, local memory, and a floating-point chip. In the Splash implementation, a processing element is composed of two modules: a character comparator and a finite state machine.

The second system that we present is from B. Schmidt et al. [15]. This system targets high performance protein database scanning on novel massively parallel architectures to gain supercomputer power at low cost. The first architecture is built around a PC cluster linked by a high-speed network and fine-grained parallel Systola 1024 processor boards connected to each node. The second architecture is the Fuzion 150, a parallel computer with a linear single-instruction, multiple-data stream (SIMD) array of 1536 processing elements on a single chip. The authors presented the design of a database scanning application based on the SW algorithm in order to derive efficient mappings onto these architectures. The implementations lead to significant runtime savings for large-scale database scanning [15].

The third system is the UCSC Kestrel parallel processor [11]. It is a single-board coprocessor with a 512-element linear

array of 8-bit SIMD processing elements. The system was designed and built at the University of California at Santa Cruz in 1993-1995, where bioinformatics applications motivated development of a sequence analysis engine that could efficiently analyze genomic databases. The architectural decisions surrounding Kestrel led to a particularly high density of computation that enables the single-board system with 9 million transistors of custom VLSI, an FPGA, and various memory chips, to outperform a current workstation 10 years later [11].

The fourth system we discuss is the S. Smith and J. Frenzel single-chip shared-memory multiprocessor architecture [10]. The principle in this architecture is to put many small and simple processors on a large integrated circuit. A system bus allows data to pass between the processors and a shared memory. For the SW algorithm application, this shared memory system may only be a cache for a much larger memory located off the integrated circuit. The database items are only read once and each result is only written once, so traffic between off-chip memory and on-chip shared memory cache is low.

The fifth and last system in this state of the art review is the Nallatech-Configurable Multi-FPGA System [7]. It is a three-FPGA network board comprising a BenNUEY motherboard with a Virtex-II XC2V3000-4 FPGA and an attached BenBLUE-II daughterboard with two Virtex-II XC2V6000-4 FPGAs. In this system the SW algorithm is implemented providing a computational speed improvement of more than two orders of magnitude (200X) [7]. The query sequences are hard-coded into the FPGAs while the individual database sequence files are loaded in the main memory. The database sequences are first written across the DIMEtalk network into the 16K read-FIFOs of the individual FPGAs. A single XC2V6000 device is capable of as many as 1.26 trillion cell updates per second. At the end of the processing, the final edit distance is written into a write-FIFO.

The performance of sequences comparison systems is commonly measured on millions of processing element or cell updates per second (MCUPS) [16]. Table 1 gives a summary of the systems mentioned in this paper, regarding their performance, design approaches, and their different technological classification [10][15].

Table 1. Performance of architectures applying the SW algorithm

Platform	Year	Technology	PEs	MCUPS
Splash II	1993	1000 nm FPGA	1536	3000?
Kestrel	1997	500 nm ASIC	512	400
Fusion 150	2000	250 nm ASIC	1536	2500
DeCypher	2001	180 nm FPGA	N/A	7500
GeneMatcher2	2001	130 nm ASIC	2872	16000

With the exception of the first architecture, the comparison of systems characteristics given in Table 1 shows a progressive increase in MCUPS matching improvements in process resolution for ASICs and FPGAs. Overall system performance is very close between these two technologies, but the great inconvenient of ASICs is their very high design costs, when FPGA systems are cheap and reconfigurable.

4. Future directions

The use of accelerators with an aim of increasing the execution speed of specific functionalities is not new, but the specific challenges of high performance calculation in bioinformatics require new solutions. The reduction of power consumption is also an important consideration. New technological evolutions highlight the importance of discharging general-purpose processors from tasks with great computational density, such as those related to biological databases.

General-purpose processors cannot respond efficiently to these challenges by themselves. In recent years, the increase in clock frequency of general-purpose processors has slowed down [17], but the transistor integration density has not. This creates two reasons explaining in part the trend towards the use of processing accelerators. The first is the fact that the accelerator can be coupled and integrated on chips along with a CPU in order to have the same technology advances such as low power consumption and best space deployment. The second and main advantage is that the integration level progression implies a great density of processing elements and a very high effective combined frequency of operation.

For instance, for CMOS technology, the International Technology Roadmap for Semiconductors identifies process resolution of 130 nm, 70 nm, and 22 nm for the years 2003, 2006 and 2016, respectively [8]. The corresponding number of processing elements that could be implemented for the application of S. Smith [10] would be 227, 782 and 5,778. Taking into account the corresponding increases in operating frequency, the equivalent combined processing frequencies are 57, 644 and 24300 GHZ. These high processing frequencies can be divided by the clock cycles per database character rate to measure the performance of systems by millions of dynamic programming cell updates per second (MCUPS).

The integration of such a large number of processing elements for the implementation of different algorithms requires sophisticated design techniques and system management.

The potential of FPGAs as reconfigurable computing accelerators arises from the fine-grained parallelism required for each incremental compute performance increase [18]. With that extra efficiency should come higher computational throughput. FPGA performance is increasing by 4× every two years, and, for operations that use architectural improvements, performance is increasing at a rate of 5X every two years [19]. Thus, FPGAs have the potential to offer a global performance increase of 20× every 2 years. Human genetic database contents are doubling every six months [9], a 16× growth over two years, so it appears that FPGA technology improvements can respond to the human genetic data explosion challenge.

For reconfigurable computing, a considerable design methodology definition effort remains to be done, because the

middle design levels are missing between the logic structures and Medium Scale Integration levels and the applications layer [18]. There is not yet the equivalent of BIOS for an FPGA-based reconfigurable computer that would isolate the OS from the particulars of each specific hardware configuration. This would be useful for supporting the development of any future OS-like layer that might run on top of an FPGA-based reconfigurable processor [18].

5. Conclusion

In this paper we presented bioinformatics applications trends, the associated high performance computational challenges, and algorithms that require efficient computing systems. We provided a classification of the existing analysis approaches, a state of the art of existing systems in this domain, and we explained accelerator trends. We observed that the preferred way to improve system performance is by increasing the global combined frequency. Using more processing elements by accelerator most effectively does this. This depends on the integration level and the optimal technical design.

With a comparison study, we observed that FPGA performances are similar those of ASICs, which have a very high cost. In addition to their affordability, FPGAs offer performance that can respond to the explosive growth in biological data and the associated future challenges. However, a significant design effort remains for system level development.

References

[1] D.W. Mount, *BIOINFORMATICS: SEQUENCE AND GENOME ANALYSIS*, Second Edition. Cold Spring Harbor Laboratory Press, New York, 2004.

[2] National Center for Biotechnology Information. BLAST home page. www.ncbi.nlm.nih.gov/blast, Jan. 2006.

[3] European Bioinformatics Institute Home Page, www.ebi.ac.uk/fasta33/, "FASTA Searching program", Jan. 2006.

[4] N.C. Jones and P.A. Pevzner, *AN INTRODUCTION TO BIOINFORMATCS ALGORITHMS*. MIT Press, Cambridge, Massachusetts London, England, 2004.

[5] Sencel's search software, www.sencel.com, Jan. 2006.

[6] C.W. Yu, K.H. Kwong, K.H. Lee and P.H.W. Leong, "A Smith-Waterman systolic cell," *Proceedings of the Tenth International Workshop on Field Programmable Logic and Applications (FPL'03)*, pp. 375-384, 2003.

[7] K. Regester, J-H. Byun, A. Mukherjee, and A. Ravindran, "Implementing bioinformatics algorithms on Nallatech-configurable multi-FPGA systems," *Xcell Journal Online*, March 2005.

[8] International Technology Roadmap for Semiconductors, *PROCESS INTEGRATION, DEVICES, AND STRUCTURES*, 2003 Edition.

[9] GenBank growth statistics, www.ncbi.nlm.nih.gov, Feb. 2005.

[10] S.F. Smith and J.F. Frenzel, "Bioinformatics searches using a single-chip shared-memory multiprocessor," *International Conference on Parallel and Distributed Processing Techniques and Applications*, 2003.

[11] A.D. Blas et al., "The UCSC Kestrel parallel processor," *IEEE Transaction on Parallel and Distributed Systems*, Vol. 16, N. 1, Jan. 2005.

[12] Paracel, Inc., www.paracel.com, Oct. 2005.

[13] TimeLogic Corp., www.timelogic.com, Jan. 2006.

[14] M. Gokhale, W. Holmes, A. Kopser, S. Lucas, R. Minnich, D. Sweely, and D. Lopresti, "Building and using a highly parallel programmable logic array," Computer, vol. 24, pp. 81-89, Jan. 1991.

[15] B. Schmidt, H. Schroder, and M. Schimmler, "Massively parallel solutions for molecular sequence analysis," *Proc. Int'l Parallel and Distributed Processing Symp.*, pp. 186-192, Apr. 2002.

[16] R. Hughey, "Parallel Sequence Comparison and Alignment," *IEEE Computer Society*, Jul. 1995.

[17] C. Johnson, J. Welser, "Future processors: flexible and modular," *3rd IEEE International Conference on Hardware Software Codesign*, Sept. 2005.

[18] K. Morris, "Saving Supercomputing with FPGAs," *FPGA and Structured ASIC Journal*, Nov. 2005.

[19] A. Schnore, M. Devlin, "SC05 OpenFPGA BOF", www.Openfpga.com, Nov. 2005.

Annexe 2. Article publié, NEWCAS, 5-8 Août 2007 à Montréal

substitution operations between the two sequences. A substitution amounts to making a deletion and an insertion (figure 1.a). The result of performing these operations allows deducing a global alignment (figure 1.b).

D(i,j)	N(j)	T	T	A	C
M(i)	0	1	2	3	4
T	1		1	2	3
T	2	1			2
C	3	2	1	2	
G	4	3	2	3	

TTAC-
|||
TT- CG

Figure 1. (a) ED calculation matrix (b) Global alignment

Calculating the score d of one cell requires its neighbors' values: diagonal a, vertical b, and horizontal c. Using the biological score of equations system (3), equation (2) takes the following form [11]:

$$d = \min \begin{cases} b + 1 \\ c + 1 \\ \begin{cases} a \rightarrow if\ (m_i = n_j) \\ a + 2 \rightarrow if\ (m_i \neq n_j) \end{cases} \end{cases} \quad (4)$$

The ED calculation has the very interesting property that the values of the horizontal and vertical neighbours (c and b) in the scores matrix differ from ± 1 with the diagonal neighbour ($a \pm 1$, $d \pm 1$) [11]. Figure (2) illustrates a general presentation of the values in this matrix:

D (i, j)	N(j)	..	n_j	..
M(i)		..	j	..
..	..	a	$b=$ $a \pm 1, d \pm 1$	
m_i	i	$c=$ $a \pm 1, d \pm 1$	d	
..	..			$\ddots$

Figure 2. Neighborhood of cell d in the scores matrix

Exploiting this property, Lipton and Lopresti simplified the calculation of the score d in any cell of the scores matrix [11]:

$$d = \begin{bmatrix} a \\ a + 2 \end{bmatrix}$$
$$if\ (b = a - 1)\ AND\ (c = a - 1)\ OR\ (m_i = n_j) \quad (5)$$
$$if\ (b = a + 1)\ AND\ (c = a + 1)\ AND\ (m_i \neq n_j)$$

In several articles presenting hardware implementations of the ED calculation [12][13][14][15][16][17], the authors discuss the SW algorithm, whereas this algorithm is used to find local alignment. In this case, the SW algorithm is guaranteed to find the best areas alignment (subsequences) of the objective and target sequences, having a maximum of similarity in the two compared sequences [9]. Its predecessor, the NW algorithm, gives a final score in the last cell of the scores matrix. A global alignment is obtained by doing a trace back.

The two algorithms use different score computation equations and produce different results. The SW algorithm includes negative scores for characters mismatches. When the DP score becomes negative, it is re-initialized to zero. This stops the progressing alignment and begins a new local alignment. The maximum score determines the beginning of the optimal local alignment zone between the two compared sequences. Thus, for two sequences M and N, the SW algorithm equations are as follows [18]:

$$S_{i,j} = \max \begin{cases} S_{i-1,j-1} + sub(N(i), M(j)) \\ S_{i,j-1} + del(N(i)) \\ S_{i-1,j} + ins(M(j)) \\ 0 \end{cases} \quad (6)$$

where $S_{i,j}$ is the comparison score for sequences N and M in position (i, j), $sub(N(i), M(j))$ is the score to align the characters in positions (i, j), and $del(N(i))$ and $ins(M(j))$ are the penalties for deletions and insertions, respectively, taking negative values [9].

In this manner, applying equation (6) replaces all the negative scores by zeros in the first line and the first column, and in all the other negative scores matrix cells. Figure 3(a) gives an example of scores calculation of using the SW algorithm [18]. The resulting local alignment is shown in figure 3(b).

D(i,j)	N(j)	T	T	G	G	A	T	T
M(i)								
A		0	0	0	0	0	0	0
C		0		0	0	0	0	0
G		0	0		2	1	0	0
G		0	0	2			2	1
T		0	2	1	3	3		4
C		0	1	1	2	2	4	4
A		0	0	0	1	4	3	3

GG- T
|||
GGAT

Figure 3. (a) SW scores matrix (b) Local alignment

3. FPGA hardware implementation considerations

To satisfy the genetic data bases size explosion challenge, the most important criterion is the processing speed improvement. This is closely related to the hardware cost, since less hardware complexity can decrease signal propagation time. It can reduce the processing period while allowing finer granularity and the implementation of more PEs. This parameter directly represents the maximum number of characters, or the compared sequences' lengths,

which can be compared in parallel. The speed performance depends on the complexity level of the PE; a simpler PE implies its improvement.

The basic comparator calculates the value of *d* according to the value of its neighbours *a*, *b*, and *c*, as given by equation (5). In this last equation, all the variables (*a*, *b*, *c* and *d*) can have a binary value [11]. So, we can use only the least significant bit of each variable to transmit the differentiations between these variables along the PEs of the SA. The binary output signal (*d_out*) of the last element drives a counter's up/down input. The counter is initialized by the objective sequence length, and is incremented or decremented according to the output signal (*d_out*) value (Fig. 4).

The data dependence imposes the neighbours' values availability: *a*, *b*, and *c*, on each level. This does not prevent a calculation parallelization of the opposite diagonal scores matrix cells. This data dependence requires PE connections from predecessor towards successor. The *PE_1×1* uniform structure facilitates the SA elements interconnection in cascade, by connecting the ports *d_in/d_out*, *S_in/S_out* and *T_in/T_out* of each PE with its successor. All PEs are also fed by control and clock signals (figure 4).

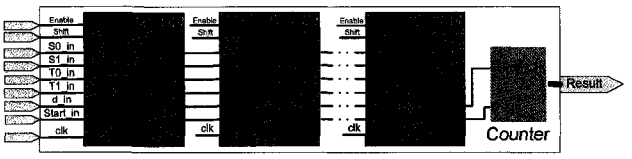


Figure 4. SA's PEs connection

To simplify the basic comparator, *a* always equals zero and is taken as a reference. The value of *b_out* is equal to 1 if *d* and *c* are same. The value of *d_out* is equal to 1 if *d* and *b* are same also. Otherwise, *b_out* and *d_out* take zero value. These two conditions allow transmitting the differentiations between the values of variables *b*, *c* and *d* of the successive PEs. We formulate these conditions and the two characters comparison result (*Compar*) by the system of equations (7), given below:

$$\begin{cases} Compar = Eq \quad AND \quad (c\_in \quad AND \quad b\_in) \\ b\_out = Compar \quad XNOR \quad c\_in \\ d\_out = Compar \quad XNOR \quad b\_in \end{cases} \tag{7}$$

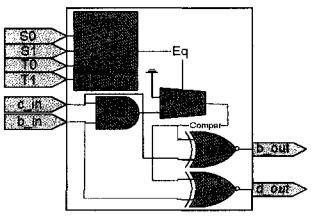


Figure 5. Elementary comparison unit

where *Eq* is a two-nucleotides n_i and m_j comparison result, selecting the output value of the multiplexer Mux_2 (*Compar*). This last value determines those of *c_out* and *b_out* according to *c_in* and *b_in*. Figure 5 implements the system of equations (7) with logic gates.

4. PE design simplification with truth tables

Figure 5 shows that the comparator is made up of three logic levels. In order to simplify its design, we described its behavior and performed an exhaustive simulation. The resulting truth table resulted in two new logical equations for *c_out* and *b_out* according to the inputs *c_in* and *b_in*:

$$\begin{cases} b\_out = \overline{Eq} \cdot c\_in + \overline{b\_in} \\ d\_out = \overline{Eq} \cdot b\_in + \overline{c\_in} \end{cases} \tag{8}$$

The resulting comparator, shown in figure 6, is composed of two logic levels only, and the synthesis report shows a signals time propagation reduction of 30%.

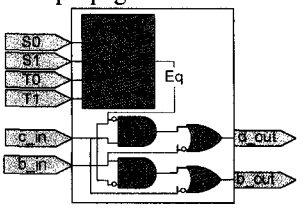


Figure 6. Comparison Unite based on truth tables

A PE is composed of one two-logic-levels comparator. The first output signal (*b_out*) is connected to the first input signal (*b_in*) of the same PE, after saving it in a flip flop for the next clock cycle (fig. 7). The second output signal (*d_out*) is also stored for the next PE's input variable *d_in* (fig. 4). This novel PE (*PE_1×1*) occupies 3 slices and 7 flip flops in Virtex I and II FPGA families, whereas the costs for previous work is 4 slices and 8 flip-flops [12]. This significant 25% reduction allows a proportional increase in the length of sequences that can be compared on-chip. In this design, each PE calculates column elements of the scores matrix, and the SA calculates its diagonal elements scores.

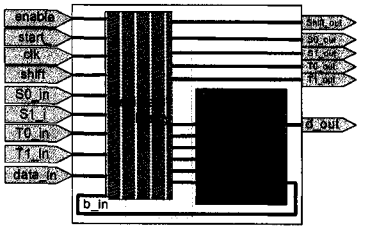


Figure 7. PE_1×1 internal structure

5. PE_1×1 Performance comparison

In this work of design and realization, we aimed at obtaining an improvement in processing acceleration and hardware cost reduction. Logic considerations (section 3)

led to the simplification of the basic comparator, on which the PEs depend. Consequently, the hardware cost and the processing speed were positively affected.

A comparative study with previous work quantifies the performance gain, and the results are shown in table 1. SC system performance is normally measured in billions of DP cell updates per second (GCUPS). Another performance parameter is the number of thousands of total EDs per second (KDPS). table 1 includes relative CUPS performance for two sequence lengths of 2700 and 7400 nucleotides for different FPGA families. The performance improvement of the proposed PE varies between 13% and 142% over previous work. The table data also illustrates the data processing load that can be taken by each design. For example, the proposed *PE_1×1×7400* is able to compare 3.54×10^6 sequences having a length of 7400 nucleotides in one minute.

System	Sequence Length	FPGA Family	Freq. (MHz)	GCUPS	KDPS	Our Speed-Up
JBits [13]	2700	xcv1000-6	186	502	69	+13%
Yu et al. [12]	2700	xcv1000-6	184	497	68	+14%
this work	2700	xcv1000-6	210	567	78	-
JBits [13]	7400	xcv6000-5	293	2168	49	+48%
HokieGene [14]	7400	xc2v6000-4	180	1332	24	+142%
this work	7400	xc2v6000-6	435	3219	59	-

Table 1. Performance comparisons

These results demonstrate that our SA design compares favorably with previous work. The proposed simplified logic equations result in significant hardware simplifications and considerable processing acceleration.

6. Conclusion

In this work, we drew up a design study based on the DP score simplification and gave precise details on the achieved results. Our objective was to refine the PE design responsible for the DP calculation.

We used a method based on the propagation of differentiation between the values of the preceding score towards the following PE which calculates the next score along the SA. We established two conditions determining values to pass to the neighbors: horizontal and vertical of the scores matrix. With the simulation results of this comparator, we obtained two simpler conditions using logical considerations. The new basic comparator composes our new PE that is the core of the SA.

Our two new logical equations produce hardware simplifications and a considerable acceleration of the SA processing frequency. The speed gain varies between 13% and 142% compared to existing similar systems.

References

[1] K. Morris, "Saving Supercomputing with FPGAs," *FPGA and Structured ASIC Journal,* Nov. 2005.

[2] C. Johnson, and J. Welser, "Future processors: flexible and modular," *3rd IEEE International Conference on Hardware Software Codesign,* Sept. 2005.

[3] GenBank growth statistics, http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html, Feb. 2007.

[4] R. Durbin et al., *BIOLOGICAL SEQUENCE ANALYSIS,* Cambridge University Press, 2003.

[5] National Center for Biotechnology Information. BLAST home page. www.ncbi.nlm.nih.gov/blast, Feb. 2007.

[6] European Bioinformatics Institute Home Page, www.ebi.ac.uk/fasta33/, "FASTA Searching program", Feb. 2007.

[7] D. Guesfield, *ALGORITHMS ON STRINGS, TREES, AND SEQUENCES,* Cambridge University Press, 1999.

[8] N.C. Jones and P.A. Pevzner, *AN INTRODUCTION TO BIOINFORMATCS ALGORITHMS.* MIT Press, Cambridge, Massachusetts London, England, 2004.

[9] D.W. Mount, *BIOINFORMATICS: SEQUENCE AND GENOME ANALYSIS,* Second Edition. Cold Spring Harbor Laboratory Press, New York, 2004.

[10] M. B. Gokhale and P. S. Graham, *RECONFIGURABLE COMPUTING: ACCELERATING COMPUTING WITH FPGA,* Springer, Dordrecht, The Netherlands, 2005.

[11] R. Lipton and D. Lopresti, "A systolic array for rapid string comparison," *Hill Conference on VLSI,* 1985.

[12] C.W. Yu, et al., "A Smith-Waterman systolic cell," *Proceedings of the Tenth International Workshop on Field Programmable Logic and Applications (FPL'03),* pp. 375-384, 2003.

[13] S. Guccione and E. Keller, "Gene matching using Jbits," *Proceedings of Field-Programmable Logic and Application Conference,* pp. 1168-1161, 2002.

[14] K. Puttegowda et al., "A run-time reconfigurable system for gene-sequence searching", Proceedings of *16th International Conference on VLSI Design,* 2003.

[15] K. Regester, et al., "Implementing bioinformatics algorithms on Nallatech-configurable multi-FPGA systems," *Xcell Journal Online,* March 2005.

[16] M. Gokhale, et al. "Building and using a highly parallel programmable logic array," *Computer,* vol. 24, pp. 81-89, Jan. 1991.

[17] B. Schmidt, et al., "A run-time reconfigurable system for gene-sequence searching", *16th International Conference on VLSI Design,* 2003.

[18] T. Smith and M. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology,* vol. 147, pp. 195-197, 1981.

[19] Basic-Algorithms-of-Bioinformatics Applet, http://www.iro.umontreal.ca/~casagran/baba.html, Département d'informatique et de recherche opérationnelle, Université de Montréal, March 2007.